Livepeer contest

2022-02-11

Overview

About C4

Code4rena (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 code contest is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the code contest outlined in this document, C4 conducted an analysis of Livepeer contest smart contract system written in Solidity. The code contest took place between January 13—January 19 2022.

Wardens

31 Wardens contributed reports to the Livepeer contest:

1. WatchPug (jtp and ming)
2. gzeon
3. harleythedog
4. Ruhum
5. Dravee
6. 0x1f8b
7. cccz
8. defsec
9. sirhashalot
10. hyh
11. danb
12. robee
13. pauliax
14. Tomio
15. kemmio
16. egjlmn1
17. Jujic

18. byterocket (pmerkleplant and pseudorandom)
19. 0x0x0x
20. ye0lde
21. rfa
22. p4st13r4 (0xb4bb4 and 0x69e8)
23. OriDabush
24. aga7hokakological
25. 0v3rf10w
26. jayjonah8
27. SolidityScan (cyberboy and zombie)

This contest was judged by 0xleastwood.

Final report assembled by itsmetechjay and CloudEllie.

Summary

The C4 analysis yielded an aggregated total of 20 unique vulnerabilities and 82 total findings. All of the issues presented here are linked back to their original finding.

Of these vulnerabilities, 1 received a risk rating in the category of HIGH severity, 8 received a risk rating in the category of MEDIUM severity, and 11 received a risk rating in the category of LOW severity.

C4 analysis also identified 24 non-critical recommendations and 38 gas optimizations.

Scope

The code under review can be found within the C4 Livepeer contest repository, and is composed of 16 smart contracts written in the Solidity programming language and includes 1928 lines of Solidity code.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities according to a methodology based on OWASP standards.

Vulnerabilities are divided into three primary risk categories: high, medium, and low.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges

- Arithmetic
- Gas use

Further information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on the C4 website.

High Risk Findings (1)

[H-01] [WP-H5] `L1Migrator.sol#migrateETH()` does not send `bridgeMinter`'s ETH to L2 causing ETH get frozen in the contract

Submitted by WatchPug, also found by gzeon, harleythedog, and Ruhum.

Per the `arb-bridge-eth` code:

> all msg.value will deposited to callValueRefundAddress on L2

- https://github.com/OffchainLabs/arbitrum/blob/78118ba205854374ed280a27415cb62c37847f72/packages/arb-bridge-eth/contracts/bridge/Inbox.sol#L313
- https://github.com/livepeer/arbitrum-lpt-bridge/blob/ebf68d11879c2798c5ec0735411b08d0bea4f287/contracts/L1/gateway/L1ArbitrumMessenger.sol#L65-L74

```
uint256 seqNum = inbox.createRetryableTicket{value: _l1CallValue}(
    target,
    _l2CallValue,
    maxSubmissionCost,
    from,
    from,
    maxGas,
    gasPriceBid,
    data
);
```

At L308-L309, ETH held by `BridgeMinter` is withdrawn to L1Migrator:

https://github.com/livepeer/arbitrum-lpt-bridge/blob/ebf68d11879c2798c5ec0735411b08d0bea4f287/contracts/L1/gateway/L1Migrator.sol#L308-L309

```
uint256 amount = IBridgeMinter(bridgeMinterAddr)
    .withdrawETHToL1Migrator();
```

However, when calling `sendTxToL2()` the parameter `_l1CallValue` is only the `msg.value`, therefore, the ETH transferred to L2 does not include any funds from `bridgeMinter`.

https://github.com/livepeer/arbitrum-lpt-bridge/blob/ebf68d11879c2798c5ec0735411b08d0bea4f287/contracts/L1/gateway/L1Migrator.sol#L318-L327

```
sendTxToL2(
    l2MigratorAddr,
    address(this), // L2 alias of this contract will receive refunds
    msg.value,
    amount,
    _maxSubmissionCost,
    _maxGas,
    _gasPriceBid,
    ""
)
```

As a result, due to lack of funds, `call` with value = amount to `l2MigratorAddr` will always fail on L2.

Since there is no other way to send ETH to L2, all the ETH from `bridgeMinter` is now frozen in the contract.

Recommendation Change to:

```
sendTxToL2(
    l2MigratorAddr,
    address(this), // L2 alias of this contract will receive refunds
    msg.value + amount, // the `amount` withdrawn from BridgeMinter should be added
    amount,
    _maxSubmissionCost,
    _maxGas,
    _gasPriceBid,
    ""
)
```

yondonfu (Livepeer) confirmed and resolved: > Fixed in https://github.com/livepeer/arbitrum-lpt-bridge/pull/51

0xleastwood (judge) commented: > Awesome find!

Medium Risk Findings (8)

[M-01] L1Migrator.migrateLPT‘ can be used to take away protocol’s access to LPT tokens in BridgeMinter

Submitted by Ruhum, also found by gzeon and harleythedog.

Same thing as the ETH issue I reported earlier. I wasn't sure if those are supposed to be a single issue or not. The concept is the same. But, now you lose LPT tokens.

The `L1Migrator.migrateLPT()` function can be called by **anyone**. It pulls all the LPT from the `BridgeMinter` contract and starts the process of moving the funds to L2. First of all, this function is only executable once. The RetryableTicket created with the first call is the only chance of moving the funds to L2.

The attacker can call the function with parameters that make the creation of the RetryableTicket on L2 fail. Thus, the LPT sits in the L1Migrator contract with no way of moving it to L2 or anywhere else. Effectively, the funds are lost.

Proof of Concept The function is only executable once because it uses the `amount` returned by `IBridgeMinter(bridgeMinterAddr).withdrawLPTToL1Migrator()` to specify the amount of LPT to be sent to L2: https://github.com/livepeer/arbitrum-lpt-bridge/blob/main/contracts/L1/gateway/L1Migrator.sol#L342

After the first call to `migrateLPT()` that function will always return 0 since the `BridgeMinter` won't have any more LPT: https://github.com/livepeer/protocol/blob/streamflow/contracts/token/BridgeMinter.sol#L107

So after the attacker called `migrateLPT()` with insufficient funds to create a RetryableTicket on L2 we have the following state:

- BridgeMinter has 0 LPT
- L1Migrator has X amount of LPT that is not accessible. There are no functions to get the LPT out of there.
- 1 failed RetryTicket

The same thing can also be triggered by a non-malicious caller by simply providing insufficient funds. The whole design of only being able to try once is the issue here.

Recommended Mitigation Steps Instead of using the `amount` returned by `IBridgeMinter(bridgeMinterAddr).withdrawLPTToL1Migrator()` you should use the balance of the `L1Migrator` contract.

It might also make sense to **not** allow anybody to call the function. I don't see the benefit of that.

Actually, the funds aren't lost. The funds are sent to the Escrow contract which can be used to transfer the funds back to the BridgeMinter contract. Thus, you could reset the whole thing to its initial state and call `L1Migrator.migrateLPT()` again. But, a really persistent attacker has the ability to DoS the function by frontrunning any call to it which results in the RetryableTicket failing again. Thus, you'd have to transfer the funds from the Escrow contract to the BrigeMinter again and so on.

So the same scenario I've outlined earlier is still viable. It's just a bit more difficult now since it has a higher cost for the attacker now. Because of that I think it's an medium issue instead of high.

Also, the mitigation steps I've given aren't valid. You can't use the `L1Migrator` contract's balance since it will always be 0 (the funds are sent to the Escrow contract). Thus the best solution would be to just limit the access to the function.

0xleastwood (judge) commented: > Nice find! The warden has outlined a potential DOS attack which can lead to funds lost which are only recoverable by the transferring the funds in the escrow contract back to the bridge minter contract.

yondonfu (Livepeer) confirmed and resolved: > Fixed in https://github.com/livepeer/arbitrum-lpt-bridge/commit/5b6a349ad8f4e53c01d2e43eda36bbbf3037a3c9

[M-02] [WP-H3] `L1Migrator.sol#migrateETH()` Improper implementation of `L1Migrator` causing `migrateETH()` always reverts, can lead to ETH in `BridgeMinter` getting stuck in the contract

Submitted by WatchPug, also found by gzeon.

https://github.com/livepeer/arbitrum-lpt-bridge/blob/ebf68d11879c2798c5ec0735411b08d0bea4f287/contracts/L1/gateway/L1Migrator.sol#L308-L310

```
uint256 amount = IBridgeMinter(bridgeMinterAddr).withdrawETHToL1Migrator();
```

`L1Migrator.sol#migrateETH()` will call `IBridgeMinter(bridgeMinterAddr).withdrawETHToL1Migrator()` to withdraw ETH from `BridgeMinter`.

However, the current implementation of `L1Migrator` is unable to receive ETH.

https://github.com/livepeer/protocol/blob/20e7ebb86cdb4fe9285bf5fea02eb603e5d48805/contracts/token/BridgeMinter.sol#L94-L94

```
(bool ok, ) = l1MigratorAddr.call.value(address(this).balance)("");
```

A contract receiving Ether must have at least one of the functions below:

- `receive() external payable`
- `fallback() external payable`

`receive()` is called if `msg.data` is empty, otherwise `fallback()` is called.

Because `L1Migrator` implement neither `receive()` or `fallback()`, the `call` at L94 will always revert.

Impact All the ETH held by the `BridgeMinter` can get stuck in the contract.

Recommendation Add `receive() external payable {}` in `L1Migrator`.

yondonfu (Livepeer) confirmed and disagreed with severity: > Severity: 2 (Med) > > We'll fix this, but noting that the funds are recoverable because

the BridgeMinter can set a new L1Migrator that does have the receive() function which is why the suggested severity is 2 (Med).

yondonfu (Livepeer) resolved: > Fixed in https://github.com/livepeer/arbitrum-lpt-bridge/pull/50

0xleastwood (judge) commented: > Agree with sponsor, these funds are recoverable. However, the warden has identified a DOS attack, which is a valid `medium` severity issue.

[M-03] Fund loss when insufficient call value to cover fee

Submitted by gzeon.

Fund can be lost if the L1 call value provided is insufficient to cover `_maxSubmissionCost`, or stuck if insufficient to cover `_maxSubmissionCost + (_maxGas * _gasPriceBid)`.

Proof of Concept `outboundTransfer` in `L1LPTGateway` does not check if the call value is sufficient, if it is `< _maxSubmissionCost` the retryable ticket creation will fail and fund is lost; if it is `<_maxSubmissionCost + (_maxGas * _gasPriceBid)` the ticket would require manual execution.

https://github.com/livepeer/arbitrum-lpt-bridge/blob/ebf68d11879c2798c5ec0735411b08d0bea4f287/contracts/L1/gateway/L1LPTGateway.sol#L80

```
function outboundTransfer(
    address _l1Token,
    address _to,
    uint256 _amount,
    uint256 _maxGas,
    uint256 _gasPriceBid,
    bytes calldata _data
) external payable override whenNotPaused returns (bytes memory) {
    require(_l1Token == l1Lpt, "TOKEN_NOT_LPT");

    // nested scope to avoid stack too deep errors
    address from;
    uint256 seqNum;
    bytes memory extraData;
    {
        uint256 maxSubmissionCost;
        (from, maxSubmissionCost, extraData) = parseOutboundData(_data);
        require(extraData.length == 0, "CALL_HOOK_DATA_NOT_ALLOWED");

        // transfer tokens to escrow
        TokenLike(_l1Token).transferFrom(from, l1LPTEscrow, _amount);
```

```
        bytes memory outboundCalldata = getOutboundCalldata(
            _l1Token,
            from,
            _to,
            _amount,
            extraData
        );

        seqNum = sendTxToL2(
            l2Counterpart,
            from,
            maxSubmissionCost,
            _maxGas,
            _gasPriceBid,
            outboundCalldata
        );
    }

    emit DepositInitiated(_l1Token, from, _to, seqNum, _amount);

    return abi.encode(seqNum);
}
```

Recommended Mitigation Steps Add check similar to the one used in `L1GatewayRouter` provided by Arbitrum team

https://github.com/OffchainLabs/arbitrum/blob/b8366005a697000dda1f57a78a7bdb2313db8fe2/packages/arb-bridge-peripherals/contracts/tokenbridge/ethereum/gateway/L1GatewayRouter.sol#L236

```
uint256 expectedEth = _maxSubmissionCost + (_maxGas * _gasPriceBid);
require(_maxSubmissionCost > 0, "NO_SUBMISSION_COST");
require(msg.value == expectedEth, "WRONG_ETH_VALUE");
```

yondonfu (Livepeer) confirmed and disagreed with severity: > Labeled as disagree with severity because we think this is a 2 - Med finding. We think that the likelihood of this occurring is low because in almost all cases users should be interacting with this contract using an application that handles calculating the maxSubmissionCost properly which would prevent the reported issue. However, we do think that the impact *if* this occurs is high since LPT *and* ETH could be lost if the reported issue happens. Thus, we think 2 - Med is appropriate given that assets are not at direct risk, but there is a low probability path for assets to be lost.

0xleastwood (judge) commented: > I agree, there is potential for unintentional loss of funds, however, the attack vector makes assumptions on how this might occur. Due to the unlikely nature, I agree that this should be a `medium`

severity issue..

yondonfu (Livepeer) resolved: > Fixed in https://github.com/livepeer/arbitrum-lpt-bridge/commit/6615bb8ce0fced895acd0e5d3e826c1e7b5d0138

[M-04] [WP-M0] `MINTER_ROLE` can be granted by the deployer of L2LivepeerToken and mint arbitrary amount of tokens

Submitted by WatchPug.

https://github.com/livepeer/arbitrum-lpt-bridge/blob/ebf68d11879c2798c5ec0735411b08d0bea4f287/contracts/L2/token/LivepeerToken.sol#L23-L30

```
function mint(address _to, uint256 _amount)
    external
    override
    onlyRole(MINTER_ROLE)
{
    _mint(_to, _amount);
    emit Mint(_to, _amount);
}
```

Using the `mint()` function of `L2LivepeerToken`, an address with `MINTER_ROLE` can burn an arbitrary amount of tokens.

If the private key of the deployer or an address with the `MINTER_ROLE` is compromised, the attacker will be able to mint an unlimited amount of LPT tokens.

We believe this is unnecessary and poses a serious centralization risk.

Recommendation Consider removing the `MINTER_ROLE`, make the `L2LivepeerToken` only mintable by the owner, and make the L2Minter contract to be the owner and therefore the only minter.

yondonfu (Livepeer) acknowledged: > Planning on keeping the role since the L2LPTGateway needs to be given minting rights as well in addition to the L2 Minter.

[M-05] [WP-M1] `BURNER_ROLE` can burn any amount of L2LivepeerToken from an arbitrary address

Submitted by WatchPug, also found by cccz.

https://github.com/livepeer/arbitrum-lpt-bridge/blob/ebf68d11879c2798c5ec0735411b08d0bea4f287/contracts/L2/token/LivepeerToken.sol#L36-L43

```
function burn(address _from, uint256 _amount)
    external
    override
    onlyRole(BURNER_ROLE)
```

```
{
    _burn(_from, _amount);
    emit Burn(_from, _amount);
}
```

Using the `burn()` function of `L2LivepeerToken`, an address with `BURNER_ROLE` can burn an arbitrary amount of tokens from any address.

We believe this is unnecessary and poses a serious centralization risk.

A malicious or compromised `BURNER_ROLE` address can take advantage of this, burn the balance of a Uniswap pool and effectively steal almost all the funds from the liquidity pool (eg, Uniswap LPT-WETH Pool).

Recommendation Consider removing the `BURNER_ROLE` and change `burn()` function to:

```
function burn(uint256 _amount)
    external
    override
{
    _burn(msg.sender, _amount);
    emit Burn(msg.sender, _amount);
}
```

https://github.com/livepeer/arbitrum-lpt-bridge/blob/49cf5401b0514511675d781a1e29d6b0325cfe88/contracts/L2/gateway/L2LPTGateway.sol#L34-L45

`Mintable(l2Lpt).burn(from, _amount);` in `L2LPTGateway.sol#outboundTransfer()` should also be replaced with:

```
Mintable(l2Lpt).transferFrom(from, _amount);
Mintable(l2Lpt).burn(_amount);
```

yondonfu (Livepeer) confirmed and resolved: > Fixed in https://github.com/livepeer/arbitrum-lpt-bridge/pull/52

[M-06] [WP-M2] `DEFAULT_ADMIN_ROLE` can approve arbitrary address to spend any amount from the `L1Escrow` contract

Submitted by WatchPug.

https://github.com/livepeer/arbitrum-lpt-bridge/blob/ebf68d11879c2798c5ec0735411b08d0bea4f287/contracts/L1/escrow/L1Escrow.sol#L21-L28

```
function approve(
    address _token,
    address _spender,
    uint256 _value
) public onlyRole(DEFAULT_ADMIN_ROLE) {
```

```
    ApproveLike(_token).approve(_spender, _value);
    emit Approve(_token, _spender, _value);
}
```

`L1Escrow.sol#approve()` allows an address with `DEFAULT_ADMIN_ROLE` can approve an arbitrary amount of tokens to any address.

We believe this is unnecessary and poses a serious centralization risk.

A malicious or compromised `DEFAULT_ADMIN_ROLE` address can take advantage of this, and steal all the funds from the `L1Escrow` contract.

Recommendation Consider removing `approve()` function and approve `l1LPT` to `l1Gateway` in the constructor.

yondonfu (Livepeer) acknowledged: > Likely won't change as we want to preserve the ability for protocol governance to move the LPT from the L1Escrow in the event of a L2 failure.

[M-07] [WP-M4] Unable to use `L2GatewayRouter` to withdraw LPT from L2 to L1, as `L2LPTGateway` does not implement `L2GatewayRouter` expected method

Submitted by WatchPug.

Per the document: https://github.com/code-423n4/2022-01-livepeer#l2---l1-lpt-withdrawal

> The following occurs when LPT is withdrawn from L2 to L1:
>
> The user initiates a withdrawal for X LPT. This can be done in two ways: a. Call outboundTransfer() on L2GatewayRouter which will call outboundTransfer() on L2LPTGateway b. Call outboundTransfer() directly on L2LPTGateway

The method (a) described above won't work in the current implementation due to the missing interface on `L2LPTGateway`.

When initiate a withdraw from the Arbitrum Gateway Router, `L2GatewayRouter` will call `outboundTransfer(address,address,uint256,uint256,uint256,bytes)` on `ITokenGateway(gateway)`:

```
function outboundTransfer(
    address _token,
    address _to,
    uint256 _amount,
    uint256 _maxGas,
    uint256 _gasPriceBid,
    bytes calldata _data
) external payable returns (bytes memory);
```

https://github.com/OffchainLabs/arbitrum/blob/b8366005a697000dda1f57a78a7bdb2313db8fe2/packages/arb-bridge-peripherals/contracts/tokenbridge/arbitrum/gateway/L2GatewayRouter.sol#L57-L64

```
function outboundTransfer(
    address _l1Token,
    address _to,
    uint256 _amount,
    bytes calldata _data
) public payable returns (bytes memory) {
    return outboundTransfer(_l1Token, _to, _amount, 0, 0, _data);
}
```

https://github.com/OffchainLabs/arbitrum/blob/b8366005a697000dda1f57a78a7bdb2313db8fe2/packages/arb-bridge-peripherals/contracts/tokenbridge/libraries/gateway/GatewayRouter.sol#L78-L102

```
function outboundTransfer(
    address _token,
    address _to,
    uint256 _amount,
    uint256 _maxGas,
    uint256 _gasPriceBid,
    bytes calldata _data
) public payable virtual override returns (bytes memory) {
    address gateway = getGateway(_token);
    bytes memory gatewayData = GatewayMessageHandler.encodeFromRouterToGateway(
        msg.sender,
        _data
    );

    emit TransferRouted(_token, msg.sender, _to, gateway);
    return
        ITokenGateway(gateway).outboundTransfer{ value: msg.value }(
            _token,
            _to,
            _amount,
            _maxGas,
            _gasPriceBid,
            gatewayData
        );
}
```

However, `L2LPTGateway` dose not implement `outboundTransfer(address,address,uint256,uint256,uint25` but only `outboundTransfer(address,address,uint256,bytes)`:

https://github.com/livepeer/arbitrum-lpt-bridge/blob/ebf68d11879c2798c5ec0735411b08d0bea4f287/contracts/L2/gateway/L2LPTGateway.sol#L65-L89

```
function outboundTransfer(
    address _l1Token,
    address _to,
    uint256 _amount,
    bytes calldata _data
) public override whenNotPaused returns (bytes memory res) {
    // ...
}
```

Therefore, the desired feature to withdraw LPT from L2 to L1 via Arbitrum Router will not be working properly.

Recommendation Consider implementing the method used by Arbitrum Router.

See also the implementation of L2DaiGateway by arbitrum-dai-bridge: https://github.com/makerdao/arbitrum-dai-bridge/blob/master/contracts/l2/L2DaiGateway.sol#L88-L95

yondonfu (Livepeer) confirmed and resolved: > Fixed in https://github.com/livepeer/arbitrum-lpt-bridge/pull/57

[M-08] Admin can rug L2 Escrow tokens leading to reputation risk

Submitted by harleythedog.

The `L1Escrow` contract has the function `approve` that is callable by the admin to approve an arbitrary spender with an arbitrary amount (so they can steal all of the escrow's holdings if they want). Even if the admin is well intended, the contract can still be called out which would degrade the reputation of the protocol (e.g. see here: https://twitter.com/RugDocIO/status/1411732108029181960). LPT is valuable on the Ethereum mainnet, so this rug vector should be mitigated. It would be best to restrict this function's power by only allowing approvals to other trusted protocol contracts (like L1LPTGateway, which I believe uses the escrow's approval).

NOTE: Even if the admin is under a timelock, this is still an issue, as users have to wait a whole week to withdraw from L2 -> L1 due to the dispute period.

Proof of Concept See the `approve` function here

Recommended Mitigation Steps Restrict the power of this `approve` function so that the admin isn't able to steal funds. This can be accomplished by only allowing approvals to other protocol functions (instead of arbitrary approvals).

yondonfu (Livepeer) acknowledged

Low Risk Findings (11)

- [L-01] No checks around poll voting mechanism *Submitted by sirhashalot.*
- [L-02] Signature authentication bypass for ZERO address *Submitted by kemmio, also found by 0x1f8b, danb, gzeon, and pauliax.*
- [L-03] Front running in LivepeerToken *Submitted by 0x1f8b.*
- [L-04] Possible frozen gateway *Submitted by 0x1f8b.*
- [L-05] `DelegatorPool.sol#claim()` Inaccurate check of `claimedInitialStake < initialStake` *Submitted by WatchPug, also found by danb.*
- [L-06] `initialize` function can be called by everyone and front-run in `DelegatorPool.sol` *Submitted by Dravee, also found by cccz, defsec, and robee.*
- [L-07] Missing `_from` param comment on `LivepeerToken.sol:burn()` *Submitted by Dravee.*
- [L-08] In `L2Migrator.sol:finalizeMigrateDelegator()`, Account `l2Addr`'s existence should be checked before call *Submitted by Dravee.*
- [L-09] Missing setter function for l2MigratorAddr *Submitted by defsec.*
- [L-10] `L1LPTGateway`, `L2LPTGateway` should start off paused after deployed *Submitted by WatchPug, also found by Ruhum.*
- [L-11] L2Migrator.claimStake attempts fee transfer without checking its possibility *Submitted by hyh.*

Non-Critical Findings (24)

- [N-01] Missing 0 address check *Submitted by cccz, also found by 0v3rf10w, 0x1f8b, defsec, Dravee, hyh, jayjonah8, OriDabush, SolidityScan, and Tomio.*
- [N-02] ERC20 Approval Race Condition *Submitted by 0x1f8b.*
- [N-03] Lack of event *Submitted by 0x1f8b.*
- [N-04] The initialize function does not check for non-zero address and emit event *Submitted by Jujic.*
- [N-05] Missing event & timelock for critical only* functions *Submitted by cccz, also found by 0v3rf10w and WatchPug.*
- [N-06] L1Escrow.approve should be called before calling L1LPTGateway.finalizeInboundTransfer *Submitted by cccz.*
- [N-07] No check that the amount arg that is being minted is larger than 0 *Submitted by jayjonah8.*
- [N-08] burn() function does not check that _amount is larger than 0 *Submitted by jayjonah8.*
- [N-09] value argument in approve() function not required to be greater than 0 *Submitted by jayjonah8.*
- [N-10] _token arg should be from a whitelisted group to avoid use of malicious tokens *Submitted by jayjonah8.*
- [N-11] no check that _amount arg is greater than 0 in outboundTransfer() function *Submitted by jayjonah8.*

- [N-12] Prevent accidentally burning tokens *Submitted by Dravee.*
- [N-13] `Manager.sol:setController()` should be a two-step process *Submitted by Dravee, also found by pauliax.*
- [N-14] `BridgeMinter.sol:migrateToNewMinter()`'s `transferOwnership` should be a two-step process *Submitted by Dravee.*
- [N-15] Group related data into separate structs *Submitted by Dravee.*
- [N-16] Protocol uses floating pragmas *Submitted by jayjonah8, also found by hyh.*
- [N-17] MixinWrappers.batchRedeemWinningTickets doesn't check for supplied arrays length *Submitted by hyh.*
- [N-18] Migrate old balance on setToken *Submitted by pauliax.*
- [N-19] Not verified function inputs of public / external functions *Submitted by robee.*
- [N-20] L2LPTGateway descriptions to be corrected *Submitted by hyh, also found by sirhashalot.*
- [N-21] Name reuse *Submitted by sirhashalot.*
- [N-22] EIP2612 in token problematic *Submitted by sirhashalot.*
- [N-23] Don't use deprecated library functions *Submitted by byterocket.*
- [N-24] Solidity compiler versions mismatch *Submitted by robee.*

Gas Optimizations (38)

- [G-01] Cache and read storage variables from the stack can save gas *Submitted by WatchPug, also found by Dravee.*
- [G-02] Public functions to external *Submitted by robee.*
- [G-03] Setting `uint256` variables to `0` is redundant *Submitted by WatchPug, also found by 0x0x0x, defsec, Dravee, Jujic, robee, and ye0lde.*
- [G-04] Cache array length in for loops can save gas *Submitted by defsec, also found by byterocket, Dravee, robee, and WatchPug.*
- [G-05] Don't assign default values *Submitted by 0x0x0x, also found by sirhashalot.*
- [G-06] Prefix increments are cheaper than postfix increments *Submitted by robee, also found by 0x1f8b, defsec, Dravee, Jujic, OriDabush, and WatchPug.*
- [G-07] Gas optimization - remove method in DelegatorPool *Submitted by 0x1f8b.*
- [G-08] Gas: Control flow optimization in `L2LPTDataCache.sol:decreaseL2SupplyFromL1()` *Submitted by Dravee, also found by pauliax.*
- [G-09] Gas: Missing checks for non-zero transfer value calls *Submitted by Dravee.*
- [G-10] Gas: `DelegatorPool.sol:claim()`, a repetitive arithmetic operation should be cached *Submitted by Dravee.*
- [G-11] `DelegatorPool.sol#claim()` Use inline expression can save gas *Submitted by WatchPug, also found by Dravee.*
- [G-12] Gas: `L2LPTDataCache.sol:l1CirculatingSupply()`, strict com-

parison can avoid expensive operation *Submitted by Dravee, also found by WatchPug.*

- [G-13] Gas: `L2LPTDataCache.sol:l1CirculatingSupply()`, Storage variables should be cached *Submitted by Dravee, also found by Jujic.*
- [G-14] Gas: Mark functions as payable when users can't mistakenly send ETH *Submitted by Dravee.*
- [G-15] Gas: use `msg.sender` instead of OpenZeppelin's `_msgSender()` when GSN capabilities aren't used *Submitted by Dravee, also found by defsec and rfa.*
- [G-16] Gas: Use `msg.sender` directly instead of caching it in a variable *Submitted by Dravee.*
- [G-17] Upgrade pragma to at least 0.8.4 *Submitted by robee, also found by defsec, Dravee, and gzeon.*
- [G-18] Constant variables using keccak can be immutable *Submitted by ye0lde, also found by byterocket, defsec, Dravee, Jujic, p4st13r4, and pauliax.*
- [G-19] Revert string > 32 bytes *Submitted by sirhashalot, also found by aga7hokakological, defsec, Dravee, gzeon, Jujic, p4st13r4, robee, and WatchPug.*
- [G-20] Gas: Use `calldata` instead of `memory` for external functions where the function argument is read-only. *Submitted by Dravee, also found by defsec.*
- [G-21] Gas: Internal functions can be private if the contract is not herited *Submitted by Dravee.*
- [G-22] Gas: Consider making some constants as non-public to save gas *Submitted by Dravee.*
- [G-23] Named return issue *Submitted by robee, also found by Dravee, WatchPug, and ye0lde.*
- [G-24] Save Gas With The Unchecked Keyword (L2LPTDataCache.sol) *Submitted by ye0lde, also found by defsec, Dravee, gzeon, Jujic, and WatchPug.*
- [G-25] Drop the token parameter from since it can only be the same value *Submitted by Ruhum.*
- [G-26] Avoiding unnecessary repeated account balance read can save gas *Submitted by WatchPug, also found by gzeon, hyh, and Tomio.*
- [G-27] using empty String which is already default 0x *Submitted by Tomio.*
- [G-28] double storage call in function decreaseL2SupplyFromL1 *Submitted by Tomio.*
- [G-29] Check of `!migratedDelegators[delegator]` can be done earlier to save gas *Submitted by WatchPug.*
- [G-30] Changing bool to uint256 can save gas *Submitted by WatchPug.*
- [G-31] Inline unnecessary internal function can make the code simpler and save some gas *Submitted by WatchPug.*
- [G-32] Using immutable variables rather than local variables is cheaper *Submitted by WatchPug.*
- [G-33] Remove redundant `_setRoleAdmin()` can save gas *Submitted by*

WatchPug, also found by byterocket.

- [G-34] Unnecessary checked arithmetic in for loops *Submitted by WatchPug.*
- [G-35] Use abi.encodePacked for gas optimization *Submitted by defsec.*
- [G-36] redundant function argument *Submitted by egjlmn1.*
- [G-37] DelegatorPool.claim subtraction can be unchecked and done once *Submitted by hyh, also found by pauliax.*
- [G-38] Custom GOVERNOR_ROLE unnecessary *Submitted by sirhashalot.*

Disclosures

C4 is an open organization governed by participants in the community.

C4 Contests incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Contest submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct formal verification regarding the provided code but instead provides final verification.

C4 does not provide any guarantee or warranty regarding the security of this project. All smart contract software should be used at the sole risk and responsibility of users.