Overlay Protocol contest Findings & Analysis Report

2022-01-13

Table of contents

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 Overlay Protocol contest smart contract system written in Solidity. The code contest took place between November 16—November 22 2021.

Wardens

18 Wardens contributed reports to the Overlay Protocol contest:

1. cmichel
2. pauliax
3. hubble
4. defsec
5. harleythedog
6. gpersoon

7. WatchPug (jtp and ming)
8. xYrYuYx
9. hyh
10. gzeon
11. nathaniel
12. MetaOxNull
13. jayjonah8
14. pants
15. yeOlde
16. 0x0x0x
17. Ruhum
18. TomFrenchBlockchain

This contest was judged by LSDan (ElasticDAO).

Final report assembled by itsmetechjay and CloudEllie.

Summary

The C4 analysis yielded an aggregated total of 25 unique vulnerabilities and 68 total findings. All of the issues presented here are linked back to their original finding.

Of these vulnerabilities, 2 received a risk rating in the category of HIGH severity, 9 received a risk rating in the category of MEDIUM severity, and 14 received a risk rating in the category of LOW severity.

C4 analysis also identified 7 non-critical recommendations and 36 gas optimizations.

Scope

The code under review can be found within the C4 Overlay Protocol contest repository, and is composed of 58 smart contracts written in the Solidity programming language and includes 6155 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 (2)

[H-01] `OverlayV1UniswapV3Market` computes wrong market liquidity

Submitted by cmichel

The `OverlayV1UniswapV3Market.fetchPricePoint` tries to compute the market depth in OVL terms as `marketLiquidity (in ETH) / ovlPrice (in ETH per OVL)`. To get the market liquidity *in ETH* (and not the other token pair), it uses the `ethIs0` boolean.

```
_marketLiquidity = ethIs0
    ? ( uint256(_liquidity) << 96 ) / _sqrtPrice
    : FullMath.mulDiv(uint256(_liquidity), _sqrtPrice, X96);
```

However, `ethIs0` boolean refers to the `ovlFeed`, whereas the `_liquidity` refers to the `marketFeed`, and therefore the `ethIs0` boolean has nothing to do with the *market* feed where the liquidity is taken from:

```
// in constructor, if token0 is eth refers to ovlFeed
ethIs0 = IUniswapV3Pool(_ovlFeed).token0() == _eth;

// in fetchPricePoint, _liquidity comes from different market fe
( _ticks, _liqs ) = IUniswapV3Pool(marketFeed).observe(_secondsA
_marketLiquidity = ethIs0
    ? ( uint256(_liquidity) << 96 ) / _sqrtPrice
    : FullMath.mulDiv(uint256(_liquidity), _sqrtPrice, X96);
```

Impact

If the `ovlFeed` and `marketFeed` do not have the same token position for the ETH pair (ETH is either token 0 or token 1 for **both** pairs), then the market liquidity & depth is computed wrong (inverted). For example, the `OverlayV1Market.depth()` function will return a wrong depth which is used in the market cap computation.

Recommended Mitigation Steps

It seems that `marketFeed.token0() == WETH` should be used in `fetchPricePoint` to compute the liquidity instead of `ovlFeed.token0() == WETH`.

realisation (Overlay) confirmed:

> Yeah, was aware of this, just hadn't finalized it in the code as of yet.

[H-02] OZ ERC1155Supply vulnerability

Submitted by pauliax, also found by hubble and defsec

Impact

Overlay uses OZ contracts version 4.3.2:

```
dependencies:
  - OpenZeppelin/openzeppelin-contracts@4.3.2
```

and has a contract that inherits from ERC1155Supply:

```
contract OverlayV1OVLCollateral is ERC1155Supply
```

This version has a recently discovered vulnerability: https://github.com/OpenZeppelin/openzeppelin-contracts/security/advisories/GHSA-wmpv-c2jp-j2xg

In your case, function unwind relies on totalSupply when calculating `\_userNotional`, `\_userDebt`, `\_userCost`, and `\_userOi`, so a malicious actor can exploit this vulnerability by first calling 'build' and then on callback 'unwind' in the same transaction before the total supply is updated.

Recommended Mitigation Steps

Consider updating to a patched version of 4.3.3.

mikeyrf (Overlay) confirmed

Medium Risk Findings (9)

[M-01] isUnderwater returns opposite boolean for short positions

Submitted by harleythedog

Impact

The function `isUnderwater` should return true if the position value is < 0. In the case of a short position, this is when oi * (2 - priceFrame) - debt < 0 (based on the logic given in the _value function). Rearranging this equation, a short position is underwater if oi * 2 < oi * priceFrame + debt. However, in the function `\_isUnderwater` in Position.sol, the left and right side of this equation is flipped,

meaning that the function will return the opposite of what it should when called on short positions.

Fortunately, the V1 implementation of `OverlayOVLCollateral` does not directly use the `isUnderwater` function in major control flow changes. However, line 304 of OverlayV1OVLCollateral.sol is a comment that says:

// TODO: think through edge case of underwater position ... and fee adjustments ...

which hints that this function is going to be used to deal with underwater positions. As a result, this issue would have a huge impact if not properly dealt with.

Proof of Concept

See code for `\_isUnderwater` here: https://github.com/code-423n4/2021-11-overlay/blob/1833b792caf3eb8756b1ba5f50f9c2ce085e54d0/contracts/libraries/Position.sol#L70

Notice that for short positions the inequality is flipped from what it should be (indeed, when self.debt is higher it is more likely that `isUnder` will be false, which is obviously incorrect).

Also, see the TODO comment here that shows `isUnderwater` is important: https://github.com/code-423n4/2021-11-overlay/blob/1833b792caf3eb8756b1ba5f50f9c2ce085e54d0/contracts/collateral/OverlayV1OVLCollateral.sol#L304

Tools Used

Inspection

Recommended Mitigation Steps

Flip the left and right side of the inequality for short positions in `\_isUnderwater`.

mikeyrf (Overlay) disagreed with severity:

> disagree with severity - `isUnderwater()` isn't used anywhere in the collateral manager and markets. Is more for information purposes, so would rate this at a

> severity of 2 - Medium in the event we had actually used this function for something more important

dmvt (judge) commented:

> I agree with the sponsor here. This represents a severe, but hypothetical issue.

[M-02] pow() is missing check on input parameters with 0 value

Submitted by gpersoon

Impact

The contract LogExpMath.sol seems to be a fork of the balancer LogExpMath.sol contract. It is mostly similar, except for checks for x and y being 0 in the beginning of the function `pow()` , see below.

This omission might lead to unexpected results.

Proof of Concept

https://github.com/code-423n4/2021-11-overlay/blob/914bed22f190ebe7088194453bab08c424c3f70c/contracts/libraries/LogExpMath.sol#L93-L110

```
    function pow(uint256 x, uint256 y) internal pure returns (uint2
        unchecked {
        _require(x < 2**255, Errors.X_OUT_OF_BOUNDS);
```

https://github.com/balancer-labs/balancer-v2-monorepo/blob/master/pkg/solidity-utils/contracts/math/LogExpMath.sol#L93-L109

```
function pow(uint256 x, uint256 y) internal pure returns (uint25
    if (y == 0) {
        // We solve the 0^0 indetermination by making it equal o
        return uint256(ONE_18);
    }
```

```
        if (x == 0) {
            return 0;
        }
        _require(x < 2**255, Errors.X_OUT_OF_BOUNDS);
```

Recommended Mitigation Steps

Check if the extra code of the balance contract is useful and if so add it.

realisation (Overlay) disputed:

> Out of scope

dmvt (judge) commented:

> I disagree with sponsor regarding scope. The Contracts section of the Contest Scope lists several contracts which rely on `contracts/libraries/FixedPoint.sol`. This contract uses the `pow` function containing the issue described. The warden has not described an exact attack but has show a math issue, which can certainly lead to a hypothetical loss of funds. Medium severity is appropriate and sponsor should definitely fix this.

[M-03] Can't enableCollateral after a disableCollateral

Submitted by gpersoon

Impact

The function `disableCollateral` of OverlayV1Mothership.sol doesn't set `collateralActive\[\_collateral] = false;` But it does revoke the roles.

Now `enableCollateral` can never be used because `collateralActive\[\_collateral] ==true` and it will never pass the second require. So you can never grant the roles again.

Note: `enableCollateral` also doesn't set `collateralActive\[\_collateral] = true`

Proof of Concept

https://github.com/code-423n4/2021-11-overlay/blob/914bed22f190ebe7088194453bab08c424c3f70c/contracts/mothership/OverlayV1Mothership.sol#L133-L153

```
    function enableCollateral (address _collateral) external onlyGov
        require(collateralExists[_collateral], "OVLV1:!exists");
        require(!collateralActive[_collateral], "OVLV1:!disabled");
        OverlayToken(ovl).grantRole(OverlayToken(ovl).MINTER_ROLE(),
        OverlayToken(ovl).grantRole(OverlayToken(ovl).BURNER_ROLE(),
    }

    function disableCollateral (address _collateral) external onlyGo
        require(collateralActive[_collateral], "OVLV1:!enabled");
        OverlayToken(ovl).revokeRole(OverlayToken(ovl).MINTER_ROLE()
        OverlayToken(ovl).revokeRole(OverlayToken(ovl).BURNER_ROLE()
    }
```

Recommended Mitigation Steps

In function `enableCollateral()` add the following (after the require):

```
collateralActive\[\_collateral] = true;
```

In function `disableCollateral` add the following (after the require):

```
collateralActive\[\_collateral] = false;
```

mikeyrf (Overlay) confirmed

[M-04] _totalSupply not updated in _transferMint() and _transferBurn()

Submitted by gpersoon, also found by WatchPug, harleythedog, hubble, xYrYuYx, cmichel, and defsec

Impact

The functions `\_transferMint()` and `\_transferBurn()` of OverlayToken.sol don't update `\_totalSupply`. Whereas the similar functions `\_mint()` and

`\_burn()` do update `\_totalSupply`.

This means that `\_totalSupply` and `totalSupply()` will not show a realistic view of the total OVL tokens.

For the protocol itself it isn't such a problem because this value isn't used in the protocol (as far as I can see). But other protocols building on Overlay may use it, as well as user interfaces and analytic platforms.

Proof of Concept

https://github.com/code-423n4/2021-11-overlay/blob/914bed22f190ebe7088194453bab08c424c3f70c/contracts/ovl/OverlayToken.sol#L349-L364

```
    function _mint( address account, uint256 amount) internal virtua
        ...
            _totalSupply += amount;
```

https://github.com/code-423n4/2021-11-overlay/blob/914bed22f190ebe7088194453bab08c424c3f70c/contracts/ovl/OverlayToken.sol#L376-L395

```
        function _burn(address account, uint256 amount) internal vir
            ...
                _totalSupply -= amount;
```

```
212 https:
213
286 https:
```

Recommended Mitigation Steps

Update `_totalSupply` in `_transferMint()` and `_transferBurn()`

realisation (Overlay) commented:

> We're not sure if this is a 1 or a 2. Definitely, at least a one - this is an incorrect implementation of the spec.

> But is it a two? It wouldn't lose funds with our contracts, we make no use of the total supply of OVL in our accounting.

> This might prove to be a vulnerability if another protocol, like Ribbon, used us for a vault of theirs, made use of total supply, and failed to discern this problem.

[M-05] Fee double counting for underwater positions

Submitted by hyh

Impact

Actual available fees are less than recorded. That's because a part of them corresponds to underwater positions, and will not have the correct amount stored with the contract: when calculation happens the fee is recorded first, then there is a check for position health, and the funds are channeled to cover the debt firsthand. This way in a case of unfunded position the fee is recorded, but cannot be allocated, so the fees accounted can be greater than the value of fees stored.

This can lead to fee withdrawal malfunction, i.e. `disburse()` will burn more and attempt to transfer more than needed. This leads either to inability to withdraw fees when disburse is failing due to lack of funds, or funds leakage to fees and then inability to perform other withdrawals because of lack of funds.

Proof of Concept

The fees are accounted for before position health check and aren't corrected thereafter when there is a shortage of funds.

https://github.com/code-423n4/2021-11-overlay/blob/main/contracts/collateral/OverlayV1OVLCollateral.sol#L311

Recommended Mitigation Steps

Adjust fees after position health check: accrue fees only on a remaining part of position that is available after taking debt into account.

Now:

```
uint _feeAmount = _userNotional.mulUp(mothership.fee());

uint _userValueAdjusted = _userNotional - _feeAmount;
if (_userValueAdjusted > _userDebt) _userValueAdjusted -= _userI
else _userValueAdjusted = 0;
```

To be:

```
uint _feeAmount = _userNotional.mulUp(mothership.fee());

uint _userValueAdjusted = _userNotional - _feeAmount;
if (_userValueAdjusted > _userDebt) {
    _userValueAdjusted -= _userDebt;
} else {
    _userValueAdjusted = 0;
    _feeAmount = _userNotional > _userDebt ? _userNotional - _us
}
```

mikeyrf (Overlay) confirmed

[M-06] Timelock and events for governor functions

Submitted by pauliax

Impact

There are contracts that contain functions that change important parameters of the system, e.g. `OverlayV1Mothership` has `setOVL`, `initializeMarket`, `disableMarket`, `enableMarket`, `initializeCollateral`, `enableCollateral`, `disableCollateral`, `adjustGlobalParams`. None of these functions emit events, nor they are timelocked. Usually, it is a good practice to give time for users to react and adjust to changes.

A similar issue was submitted in a previous contest and assigned a severity of Medium: https://github.com/code-423n4/2021-09-swivel-findings/issues/101

Recommended Mitigation Steps

Consider using a timelock for critical params of the system and emitting events to inform the outside world.

realisation (Overlay) commented:

> The plan has been to have a timelock at some point in the protocol. Probably on whatever is the admin for the mothership. But this just had to be evaluated. It might be on the market contract itself, or on the addresses granted the role of admin.

mikeyrf (Overlay) commented:

> duplicate #64

dmvt (judge) commented:

> I'm removing the duplicate in this case because issue #64 refers exclusively to the events. This issue is focused primarily on the lack of governance timelock, which has traditionally been considered a medium severity issue.

[M-07] Cached version of ovl may be outdated

Submitted by pauliax

Impact

contract OverlayV1OVLCollateral and OverlayV1Governance cache ovl address:

```
    IOverlayTokenNew immutable public ovl;
```

This variable is initialized in the constructor and fetched from the mothership contract:

```
    mothership = IOverlayV1Mothership(_mothership);
    ovl = IOverlayV1Mothership(_mothership).ovl();
```

ovl is declared as immutable and later contract interacts with this cached version. However, mothership contains a setter function, so the governor can point it to a new address:

```
function setOVL (address _ovl) external onlyGovernor {
    ovl = _ovl;
}
```

`OverlayV1OVLCollateral` and `OverlayV1Governance` will still use this old cached value.

Recommended Mitigation Steps

Consider if this was intended, or you want to remove this cached version and always fetch on the go (this will increase the gas costs though).

realisation (Overlay) commented:

> This is just a detail we were yet to settle on but definitely were going to as we got the contracts to a totally deployable state.

mikeyrf (Overlay) disagreed with severity:

> disagree w severity reason - would put this at 1 - Low Risk given the governor would be responsible for properly setting

dmvt (judge) commented:

> I agree with the warden that this constitutes a medium risk.

> From the judging criteria (emphasis mine):

> 2 — Med (M): vulns have a risk of 2 and are considered “Medium” severity when **assets are not at direct risk, but the function of the protocol or its availability could be impacted**, or leak value with a hypothetical attack path with stated assumptions, but external requirements.

[M-08] OverlayToken.burn function could burn tokens of any user

Submitted by xYrYuYx

Impact

https://github.com/code-423n4/2021-11-overlay/blob/main/contracts/ovl/OverlayToken.sol#L366

The burner could burn any amount of tokens of any user. This is not good solution of burn

Tools Used

Manual

Recommended Mitigation Steps

Update burn function for only owner can burn his tokens. Now, `ovl.burn` function is used in OverlayV1OVLCollateral.sol file, and these updates won't make any issue in protocol.

mikeyrf (Overlay) acknowledged:

> sponsor acknowledged reason - `onlyBurner` modifier with access control privileges prevent unexpected burn amounts, given only collateral managers are given burn permissions

[M-09] Improper Upper Bound Definition on the Fee

Submitted by defsec, also found by gzeon, nathaniel, WatchPug, cmichel, and pauliax

Impact

In the `adjustGlobalParams` function on line 1603 of "https://github.com/code-423n4/2021-11-overlay/blob/main/contracts/mothership/OverlayV1Mothership.sol#L1630",

`adjustGlobalParams` function does not have any upper or lower bounds. Values that are too large will lead to reversions in several critical functions.

Proof of Concept

- The `setFee` function that begins on line 163 of `adjustGlobalParams` sets the liquidity and transaction fee rates for the market in which the function is called. In this context, the transaction fee is the percentage of a transaction that is taken by the protocol and moved to a designated reserve account. As the name suggests, transaction fees factor in to many of the essential transaction types performed within the system.
- Navigate to "https://github.com/code-423n4/2021-11-overlay/blob/main/contracts/mothership/OverlayV1Mothership.sol#L163" contract and go to line #163.
- On the function there is no upper and lower bound defined. Therefore, users can pay higher fees.

Tools Used

None

Recommended Mitigation Steps

Consider defining upper and lower bounds on the `adjustGlobalParams` function.

mikeyrf (Overlay) confirmed

dmvt (judge) commented:

> Several wardens have marked this issue as high severity due to the potential for governance to rug users. Several have marked it as low risk because it's really just a bounding issue and presumably governance would not willingly choose to rug their users.

> I view this a medium severity issue. If exploited, the impact would be high. The likelihood that it would be exploited intentionally or happen unintentionally is low, but not impossible as the uninformed users dynamic could come into play here.

Low Risk Findings (14)

- [L-01] OVL token shouldn't be available for substitution, needs to be set only once *Submitted by hyh*
- [L-02] Incorrect position indexing *Submitted by xYrYuYx*
- [L-03] OverlayV1Market.update function is public function *Submitted by xYrYuYx*
- [L-04] Constructor Lack of Input Validation for _compoundingPeriod *Submitted by MetaOxNull*
- [L-05] `OverlayToken.sol` Insufficient input validation *Submitted by WatchPug*
- [L-06] Missing setter function for `OverlayV1Mothership#marginBurnRate` *Submitted by WatchPug*
- [L-07] `OverlayV1UniswapV3Market` assumes one of the tokens is ETH *Submitted by cmichel*
- [L-08] Missing `macroWindow > microWindow` check *Submitted by cmichel*
- [L-09] contract OverlayV1OI isn't abstract *Submitted by gpersoon*
- [L-10] No user friendly error message when _leverage==0 *Submitted by gpersoon*
- [L-11] Should add reentrancy guard modifiers *Submitted by jayjonah8*
- [L-12] Open TODOs in Codebase *Submitted by pauliax, also found by MetaOxNull, pants, and yeOlde*
- [L-13] Discrepancies between the interface and implementation *Submitted by pauliax*
- [L-14] Incorrect comments *Submitted by yeOlde*

Non-Critical Findings (7)

- [N-01] approve function is vulnerable *Submitted by pants, also found by WatchPug*
- [N-02] Missing events for critical operations *Submitted by WatchPug, also found by 0x0x0x, harleythedog, defsec, Ruhum, and xYrYuYx*
- [N-03] _rewardsTo not empty *Submitted by pauliax*

- [N-04] Context and msg.sender *Submitted by pauliax, also found by WatchPug*
- [N-05] Incorrect naming issue *Submitted by xYrYuYx, also found by WatchPug*
- [N-06] Typos *Submitted by yeOlde*
- [N-07] Commented out code (no explanation) *Submitted by yeOlde*

Gas Optimizations (36)

- [G-01] OverlayV1Governance.setEverything does unnecessary function calls *Submitted by hyh*
- [G-02] Unnecessary castings in `OverlayV1UniswapV3Market.fetchPricePoint()` *Submitted by pants*
- [G-03] require should come first *Submitted by pants*
- [G-04] State variables can be `immutable` s *Submitted by pants, also found by harleythedog*
- [G-05] Dead code *Submitted by pauliax, also found by WatchPug, defsec, and gzeon*
- [G-06] Cache storage access *Submitted by pauliax*
- [G-07] Eliminate duplicate math operations *Submitted by pauliax*
- [G-08] Eliminate subtraction *Submitted by pauliax*
- [G-09] Pack structs tightly *Submitted by pauliax, also found by gzeon*
- [G-10] _beforeTokenTransfer and _afterTokenTransfer functions are empty *Submitted by xYrYuYx, also found by MetaOxNull*
- [G-11] Use `external` keyword instead of `public` for some functions *Submitted by xYrYuYx, also found by defsec*
- [G-12] Unused Named Returns *Submitted by yeOlde*
- [G-13] Unneeded variable and code in enterOI (OverlayV1Market.sol) *Submitted by yeOlde*
- [G-14] Constructor Does Not Check for Zero Addresses *Submitted by MetaOxNull, also found by hyh, and xYrYuYx*
- [G-15] Use msg.sender Rather Than _msgSender() to Save Gas *Submitted by MetaOxNull*

- [G-16] Adding unchecked directive can save gas *Submitted by WatchPug, also found by yeOlde, defsec, and harleythedog*
- [G-17] Redundant code *Submitted by WatchPug*
- [G-18] Use `transferBurn` can save gas *Submitted by WatchPug*
- [G-19] Use short reason strings can save gas *Submitted by WatchPug, also found by pants and yeOlde*
- [G-20] `OverlayToken.sol` Check of allowance can be done earlier to save gas *Submitted by WatchPug*
- [G-21] Avoiding external calls can save gas *Submitted by WatchPug*
- [G-22] Change unnecessary storage variables to constants can save gas *Submitted by WatchPug, also found by defsec*
- [G-23] `OverlayV1Market.sol#lock()` Switching between 1, 2 instead of 0, 1 is more gas efficient *Submitted by WatchPug*
- [G-24] Cache storage variables in the stack can save gas *Submitted by WatchPug*
- [G-25] At `OverlayV1Comptroller.sol`, `_roller.time` shouldn't be cached *Submitted by 0x0x0x*
- [G-26] Optimize `OverlayV10l#_oi` *Submitted by 0x0x0x*
- [G-27] `_fundingFactor` at `OverlayV10l#computeFunding` can be calculated cheaper *Submitted by 0x0x0x*
- [G-28] `OverlayV1PricePoint.sol#pricePoints` can be implemented more efficiently *Submitted by 0x0x0x*
- [G-29] Use of constant keccak variables results in extra hashing (and so gas). *Submitted by defsec, also found by pauliax*
- [G-30] `> 0` can be replaced with `!= 0` for gas optimization *Submitted by defsec*
- [G-31] Simplify function roll() *Submitted by gpersoon, also found by 0x0x0x*
- [G-32] Use _brrrrdExpected everywhere in oiCap() *Submitted by gpersoon*
- [G-33] Check for liquidation in value() *Submitted by gpersoon*
- [G-34] Use _userOiShares everywhere in unwind() *Submitted by gpersoon*

- [G-35] All overflow/underflow checks are automatic in Solidity 0.8 *Submitted by harleythedog, also found by xYrYuYx and 0x0x0x*
- [G-36] OverlayV1OVLCollateral.liquidate storage pos.market variable is read up to three times, can be saved to memory *Submitted by hyh, also found by MetaOxNull*

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.

Top