SALUS SECURITY

OCT 2023

CODE SECURITY ASSESSMENT

AQUEDUCT

Overview

Project Summary

- Name: Aqueduct - TWAMM
- Version: commit 35f120e
- Platform: EVM-compatible Chains
- Language: Solidity
- Repository:
  - https://github.com/aqueduct-finance/aqueduct-twamm
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Aqueduct - TWAMM
Version	v2
Type	Solidity
Dates	Oct 09 2023
Logs	Sep 06 2023; Oct 09 2023

Vulnerability Summary

Total High-Severity issues	2
Total Medium-Severity issues	0
Total Low-Severity issues	4
Total informational issues	5
Total	11

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Bidders can enforce their own bid in certain situations	High	Business Logic	Resolved
2	placeBid() calculates bidValue incorrectly if the current auction.token is token1	High	Business Logic	Acknowledged
3	No slippage protection	Low	Business Logic	Resolved
4	Missing reentrancy protection	Low	Reentrancy	Resolved
5	Missing events for functions that change important state	Low	Logging	Resolved
6	Unlocked pragma	Low	Configuration	Resolved
7	Redundant code	Informational	Redundancy	Resolved
8	Use of assert	Informational	Data Validation	Resolved
9	IAqueductV1Pair is missing functions present in implementation	Informational	Code Quality	Resolved
10	Potential revert may jail AqueductV1Pair	Informational	Business Logic	Partially Resolved
11	Gas optimization suggestions	Informational	Gas Optimization	Resolved

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Bidders can enforce their own bid in certain situations

Severity: High	Category: Business Logic
Target: - src/AqueductV1Auction.sol	

Description

By default if a user places a bid and there are no other bids in the same block, the user becomes the winner and can call the executeWinningBid() in the next block.

A malicious user can then place a bid in the subsequent block right before executeWinningBid(). In this case, auction.lastAuctionTimestamp will not be updated since auction.winningBid is greater than 0. The bid placed in the previous block will be executed during the execution of placeBid(). After placing the bid, the executeWinningBid() will execute the bid that the malicious user placed in the same block. In this case, the malicious user can win the auction with a low bid since there is no one competing. However, executeWinningBid() should not execute the bid from the malicious user while the auction is still in progress.

src/AqueductV1Auction.sol:L80-L144

```
function placeBid(
        address token,
        address pair,
        uint256 bid,
        uint256 swapAmount,
        uint256 deadline
) external ensure(deadline) placeBidLock {
        Auction memory auction = getAuction[pair];

        if (block.timestamp > auction.lastAuctionTimestamp) {
            // if there is a winningBid, execute previous auction, otherwise just reset
timestamp
            if (auction.winningBid > 0) {
                executeWinningBid(pair);
                auction = getAuction[pair];
            } else {
                auction.lastAuctionTimestamp = block.timestamp;
            }
        }

        ...
        // update auction
        auction.token = token;
        auction.winningBid = bid;
        auction.winningSwapAmount = swapAmount;
        auction.lockedSwapAmountOut = amountOut;
        auction.winningBidderAddress = msg.sender;
        getAuction[pair] = auction;
}
```

src/AqueductV1Auction.sol:L152:

```
function executeWinningBid(address pair) public executeWinningBidLock {
        Auction memory auction = getAuction[pair];
        if (block.timestamp <= auction.lastAuctionTimestamp || auction.winningBid == 0)
        revert AUCTION_ALREADY_EXECUTED();
```

Proof of Concept

Add the following code in test/hardhat/AqueductV1Auction.spec.ts file, and run `npx hardhat test --grep auction:place_execute_bid_in_one_block`.

In the following test, a user places the first bid. In the next block, another user places the second bid and executes the bid he placed.

```
it("auction:place_execute_bid_in_one_block", async () => {
    const { pair, wallet, token0, token1, auction, other } = await loadFixture(fixture);

    const token0Amount = expandTo18Decimals(5);
    const token1Amount = expandTo18Decimals(10);
    await addLiquidity(token0, token1, pair, wallet, token0Amount, token1Amount);

    const token1InitialBalance = BigInt(await token1.balanceOf({
        account: wallet.address,
        providerOrSigner: ethers.provider,
    }));

    // first bid
    const bid1 = expandTo18Decimals(1);
    const swapAmount = expandTo18Decimals(1);
    await token0.approve({
        receiver: auction.address,
        amount: ethers.constants.MaxInt256,
    })
    .exec(wallet);
    await auction.placeBid(token0.address, pair.address, bid1, swapAmount,
ethers.constants.MaxUint256);

    // let executeWinningBid() be in the same block with the second-placed bid
    await network.provider.send("evm_setAutomine", [false]);

    // second bid
    const bid2 = expandTo18Decimals(2);
    const swapAmount2 = expandTo18Decimals(2);
    await token0.approve({
        receiver: auction.address,
        amount: ethers.constants.MaxInt256,
    })
    .exec(other);
    await auction.connect(other)
    .placeBid(token0.address, pair.address, bid2, swapAmount2,
ethers.constants.MaxUint256);

    await network.provider.send("evm_setAutomine", [true]);

    // second bid should not be executed until the next block (current implementation of
AqueductV1Auction fails this test)
    await expect(auction.executeWinningBid(pair.address))
    .to.be.revertedWithCustomError(auction, "AUCTION_ALREADY_EXECUTED");
    // the first bid should be executed
    expect(await token1.balanceOf({
        account: wallet.address,
        providerOrSigner:ethers.provider
    })).to.equal(token1InitialBalance + BigInt("1666666666666666666"));
})
```

Recommendation

It is necessary to update auction.lastAuctionTimestamp properly during the execution of the placeBid() function.

Status

This issue has been resolved by the team.

2. placeBid() calculates bidValue incorrectly if the current auction.token is token1

Severity: High	Category: Business Logic

Target:

- src/AqueductV1Auction.sol

Description

Basically, bidValue is based on the token0's decimal whether the token is token0 or token1, while auction.winningBid records the actual value. So if the current auction.token is token1 and the two tokens' decimals are different, comparing bid amounts will be wrong.

For example, if token0 is DAIx with 10^18 reserves and token1 is USDCx with 10^6 reserves, here's a possible attack scenario:

1. John calls placeBid() with 100 USDCx as a bid amount. The current winningBid is 100 * 10^6;
2. Alice calls placeBid() with 50 DAIx as a bid amount. Alice's bidValue will be 50 * 10^18;
3. Alice will be the winner even if her bid worth is less than John's.

src/AqueductV1Auction.sol:L122-L143

```
//  if token1, need to convert to token0 denominated value
uint256 bidValue = bid;
(reserve0, reserve1, ) = IAqueductV1Pair(pair).getReserves();
if (token == token1) {
    bidValue = (bid * reserve0) / reserve1;
}

// revert if bid's value is lte to winning bid or bid is under 0.3% of total amount
if (bidValue <= auction.winningBid || bid < ((bid + swapAmount) * 3) / 1000) revert
AUCTION_INSUFFICIENT_BID();
TransferHelper.safeTransferFrom(token, msg.sender, address(this), bid + swapAmount);

...

// update auction
auction.token = token;
auction.winningBid = bid;
auction.winningSwapAmount = swapAmount;
auction.lockedSwapAmountOut = amountOut;
auction.winningBidderAddress = msg.sender;
getAuction[pair] = auction;
```

Recommendation

It is recommended to calculate bidValue based on the current auction.token's decimal.

Status

This issue has been acknowledged by the team.

3. No slippage protection

Severity: Low	Category: Business Logic
Target: - src/AqueductV1Auction.sol	

Description

There is no slippage protection implemented in the placeBid function. As a result, trades may encounter excessive slippage and potential price manipulation, such as front-running.

Here's a possible attack scenario:

1. Alice sends a transaction to call placeBid();
2. Bob frontruns Alice’s transaction and swaps in advance (suppose Bob wins the auction);
3. If the deadline is not reached, Alice’s transaction is executed and the swap will be done at a low price;
4. Bob swaps back.

Recommendation

It is recommended to add some kind of protection for users so that they receive the minimum amount expected. For example, add the amountOutMin parameter and check if the amountOut is not less than amountOutMin.

Status

This issue has been resolved by the team.

4. Missing reentrancy protection

Severity: Low	Category: Reentrancy
Target: - src/AqueductV1Pair.sol	

Description

The retrieveFunds function in the AqueductV1Pair contract lacks reentrancy protection and its implementation does not follow the check-effects-interactions pattern.

src/AqueductV1Pair.sol:L678

```
function retrieveFunds(ISuperToken _superToken) external returns (uint256
returnedBalance)
```

Recommendation

Consider adding the `lock` modifier.

Status

This issue has been resolved by the team.

5. Missing events for functions that change important state

Severity: Low	Category: Logging

Target:
- src/AqueductV1Auction.sol
- src/AqueductV1Factory.sol

Description

Events allow capturing the changed parameters so that off-chain tools/interfaces can register such changes that allow users to evaluate them. Missing events do not promote transparency and if such changes immediately affect users' perception of fairness or trustworthiness, they could exit the protocol causing a reduction in protocol users.

Throughout the Aqueduct codebase, events are lacking in the privileged setter functions (e.g. setFeeTo(), setFeeToSetter()) and auction functions (e.g. placeBid(), executeWinningBid()).

Recommendation

It is recommended to emit events for important state changes.

Status

This issue has been resolved by the team.

6. Unlocked pragma

Severity: Low	Category: Configuration
Target: - src/AqueductV1Pair.sol - src/libraries/AqueductV1Library.sol	

Description

src/AqueductV1Pair.sol:L2

```
pragma solidity ^0.8.12;
```

In AqueductV1Library, the pairFor function calculates the CREATE2 address for a pair, in which the init code hash of AqueductV1Pair is used. Since the AqueductV1Pair contract uses a floating compiler version ^0.8.12, code may compile to different bytecodes with different compiler versions which may not match the hardcoded hash in the pairFor function. Use a locked pragma statement to get a deterministic bytecode.

src/libraries/AqueductV1Library.sol:L25-L41

```
function pairFor(address factory, address tokenA, address tokenB) internal pure returns
(address pair) {
        (address token0, address token1) = sortTokens(tokenA, tokenB);
        pair = address(
        uint160(
                uint256(
                keccak256(
                        abi.encodePacked(
                        bytes1(0xff),
                        factory,
                        keccak256(abi.encodePacked(token0, token1)),

hex"6e3566401a51fa210eff400bbdd89b9fbb2b8a44dcd51cc144e7cc5fd95d1b9f" // init code hash
                        )
                )
                )
        )
        );
}
```

Recommendation

It is recommended to use a locked Solidity version.

Status

This issue has been resolved by the team.

2.3 Informational Findings

7. Redundant code

Severity: Informational	Category: Redundancy
Target: - src/interfaces/IAqueductV1Auction.sol - src/AqueductV1Auction.sol	

Description

src/interfaces/IAqueductV1Auction.sol:L6

```
error AUCTION_PAIR_DOESNT_EXIST();
```

The AUCTION_PAIR_DOESNT_EXIST error is unused.

src/AqueductV1Auction.sol:L8

```
import {ISuperfluid, ISuperToken} from
"@superfluid-finance/ethereum-contracts/contracts/interfaces/superfluid/ISuperfluid.sol"
;
```

src/AqueductV1Factory.sol:L10

```
import {TransferHelper} from "./libraries/TransferHelper.sol";
```

src/AqueductV1Router.sol:L14

```
import {IERC20} from "./interfaces/IERC20.sol";
```

The above imports are not used.

Recommendation

Consider removing the redundant code.

Status

This issue has been resolved by the team.

8. Use of assert

Severity: Informational	Category: Data Validation
Target: - src/AqueductV1Factory.sol	

Description

In the constructor of the AqueductV1Factory contract, assert is used to check whether the address of _host is zero address. As per Solidity documentation:

Assert should only be used to test for internal errors, and to check invariants. Properly functioning code should never create a Panic, not even on invalid external input. If this happens, then there is a bug in your contract which you should fix. Language analysis tools can evaluate your contract to identify the conditions and function calls which will cause a Panic.

src/AqueductV1Factory.sol:L28

```
assert(address(_host) != address(0));
```

Recommendation

It is recommended to use a require statement or custom error instead.

Status

This issue has been resolved by the team.

### 9. IAqueductV1Pair is missing functions present in implementation	
Severity: Informational	Category: Code Quality
Target: - src/interfaces/IAqueductV1Pair.sol	

Description

The IAqueductV1Pair interface is implemented by the AqueductV1Pair contract. The implementation includes extra functions not declared in IAqueductV1Pair, e.g. retrieveFunds(), getRealTimeIncomingFlowRates(), getStaticReserves().

Recommendation

Consider adding the missing functions to the corresponding interface.

Status

This issue has been resolved by the team.

10. Potential revert may jail AqueductV1Pair

Severity: Informational	Category: Business Logic
Target: - src/AqueductV1Pair.sol	

Description

Token transfer may revert with PAIR_TRANSFER_FAILED error in Super App callbacks. As suggested by the documentation, use the trycatch pattern if performing an action which interacts with other contracts inside of the callback.

Meanwhile, in the _handleCallback function, _updateAccumulators() is called which will update state variables twap0CumulativeLast and twap1CumulativeLast. However, these two state variables never decrease. Thus, the maximum value will be reached at a certain moment and all subsequent operations will fail.

src/AqueductV1Pair.sol:L451-L484

```
function _updateAccumulators(
        uint112 reserve0,
        uint112 reserve1,
        uint112 totalFlow0,
        uint112 totalFlow1,
        uint32 time
) private {
        uint32 timeElapsedSinceInputTime = time - _blockTimestampLast;

        // update cumulatives
        // assuming reserve{0,1} are real time
        if (totalFlow1 > 0) {
        twap0CumulativeLast += _getTwapCumulative(
                reserve0,
                _reserve0,
                totalFlow0,
                totalFlow1,
                timeElapsedSinceInputTime
        );
        uint112 streamedAmount0 = totalFlow0 > 0 ? totalFlow0 * timeElapsedSinceInputTime
: 0;
        _totalSwappedFunds0 += streamedAmount0 + _reserve0 - reserve0;
        }
        if (totalFlow0 > 0) {
        twap1CumulativeLast += _getTwapCumulative(
                reserve1,
                _reserve1,
                totalFlow1,
                totalFlow0,
                timeElapsedSinceInputTime
        );
        uint112 streamedAmount1 = totalFlow1 > 0 ? totalFlow1 * timeElapsedSinceInputTime
: 0;
        _totalSwappedFunds1 += streamedAmount1 + _reserve1 - reserve1;
        }
}
```

Recommendation

It is recommended to use try-catch pattern to handle potential failures.

Consider including updates to twap0CumulativeLast and twap1CumulativeLast in the unchecked block if overflow is desired.

Status

This issue has been partially resolved by the team.

11. Gas optimization suggestions

Severity: Informational	Category: Gas Optimization
Target: - src/AqueductV1Pair.sol - src/AqueductV1Auction.sol	

Description

src/AqueductV1Auction.sol:L31

```
address public override factory;
```

The state variable factory could be declared immutable since its value is fixed after deployment.

src/AqueductV1Pair.sol:L533-L539

```
uint256 _totalSupply = totalSupply;
if (_totalSupply == 0) {
    liquidity = Math.sqrt(amount0 * amount1) - MINIMUM_LIQUIDITY;
    _mint(address(0), MINIMUM_LIQUIDITY); // permanently lock the first
MINIMUM_LIQUIDITY tokens
} else {
    liquidity = calculateLiquidity(amount0, amount1, reserve0, reserve1, totalSupply);
}
```

The highlighted state variable totalSupply could be replaced with the local variable _totalSupply.

Recommendation

Consider applying the gas optimizations where needed.

Status

This issue has been resolved by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 35f120e:

File	SHA-1 hash
src/libraries/UQ112x112.sol	a4448f52d681f2cfa78bcb0adbe803c33a2ca31a
src/libraries/TransferHelper.sol	1e77ae98fd562c63de542d41aa3dfd8288993adf
src/libraries/AqueductV1Library.sol	4fb3431819b19372d1926c6e5fcdb3feaf939062
src/libraries/Math.sol	57949caa5fd17eafeab884dca647eb91dcd4434d
src/AqueductV1Auction.sol	4a9c9f7d8a93b1dca92ab9ecb70d58b0531f7d48
src/AqueductV1Factory.sol	224b51129e71878d5018c2730d652d2094879bb6
src/AqueductV1Pair.sol	827c7f24384c17fe3b9af3fc468900483dea8f99
src/AqueductV1ERC20.sol	b3887dac863a18d60d7470805722bce6ba23249e
src/interfaces/IAqueductV1Callee.sol	0977b9da4b29a2461ebdc84e88551af63a2524c1
src/interfaces/IWETH.sol	7f5816a5aca7b80fa81d4f464a9dc821f9d869b6
src/interfaces/IAqueductV1Router01.sol	20df34bfb29f0e95a941a86db3ea16d27ce7ff70
src/interfaces/IAqueductV1ERC20.sol	4de09c239f18d1baaa33e77a35d81db48618d0bc
src/interfaces/IAqueductV1Auction.sol	4a283958da55e7c868c7a985591633550d2dbbce
src/interfaces/IAqueductV1Factory.sol	c05ff3e880d01aeaacc7cef4a60ef6cfe8ed7648
src/interfaces/IAqueductV1Pair.sol	1f09100bfdaa34fccb38c7b1790a457e2b0266f6
src/interfaces/IAqueductV1Router.sol	ca38db80f66e6523e7f292e609425c9d9ea12191
src/interfaces/IERC20.sol	9d06dc48920621d73c9e601b63242b89debf1233
src/AqueductV1Router.sol	38c7e8887ef05eb5cd31bad8341d656f7ec28fc0