MOVELABS

Meshswap V3
Security Analysis Report

rev 1.0

Prepared for
Ozys

Prepared by
MOVE LABS

INTRODUCTION

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another party. Any subsequent publication of this report shall be without mandatory consent.

Name	Ozys
Website	https://meshswap.fi/
Repository	https://github.com/meshswap-fi/audit-movelabs/tree/v3
Commit	89539c39d72a01744d1d8f3f1cd7aea3a8882b08
Platform	Polygon
Network	Mainnet
Languages	Solidity
Method	Source code auditing, Automated static analysis
Approver	Aiden Hyun
Timeline	2023-05-04 ~ 2023-06-03

TABLE OF CONTENTS

PROJECT OVERVIEW

About Project

We conducted a security audit of MeshswapV3 as part of our project. We focused on the new contracts developed for transitioning from V2 to V3 and the differences between MeshswapV3 and UniswapV3. We observed several differences between MeshswapV3 and UniswapV3. These differences included variations in governance mechanisms, reward distribution methods, and proxy structure.

Period

- Overall Period
  - 2023-05-04 ~ 2023-06-03

Project Targets

- Meshswap V3
  - Repository : https://github.com/meshswap-fi/audit-movelabs/tree/v3
  - Type : Solidity
  - Platforms : Polygon Network

Revision History

Commit	Date	Comment
505cf0bf4d6711eb76e72b01e01182d1fd657bfe	2023-05-04	Initial Code
89539c39d72a01744d1d8f3f1cd7aea3a8882b08	2023-05-15	Audited Mainly
9df7aa5ba3c497ff9e55fdd4a51426bc999af40b	2023-05-18	Issue 1 Fixed
f8cd78c255735359a85507e9749c81b6a2019c70	2023-06-08	Issue 3 Fixed
1afa99735319ac5067f6d3bfeea89c4556bc2aa4	2023-06-08	Final Code (Issue 2 Fixed)

SCOPE

We were provided with a stable source code tree to review. We also reviewed each of the committed fixes.

Source code:

- Meshswap V3
    - 89539c39d72a01744d1d8f3f1cd7aea3a8882b08

```
v3
├── core
│   ├── interfaces
│   │   ├── callback
│   │   │   ├── IUniswapV3FlashCallback.sol
│   │   │   ├── IUniswapV3MintCallback.sol
│   │   │   └── IUniswapV3SwapCallback.sol
│   │   ├── IERC20Minimal.sol
│   │   ├── IUniswapV3Factory.sol
│   │   ├── IUniswapV3PoolDeployer.sol
│   │   ├── IUniswapV3Pool.sol
│   │   ├── LICENSE
│   │   └── pool
│   │       ├── IUniswapV3PoolActions.sol
│   │       ├── IUniswapV3PoolDerivedState.sol
│   │       ├── IUniswapV3PoolEvents.sol
│   │       ├── IUniswapV3PoolImmutables.sol
│   │       ├── IUniswapV3PoolOwnerActions.sol
│   │       └── IUniswapV3PoolState.sol
│   ├── libraries
│   │   ├── BitMath.sol
│   │   ├── FixedPoint128.sol
│   │   ├── FixedPoint96.sol
│   │   ├── FullMath.sol
│   │   ├── LICENSE
│   │   ├── LICENSE_MIT
│   │   ├── LiquidityMath.sol
│   │   ├── LowGasSafeMath.sol
│   │   ├── Oracle.sol
│   │   ├── Position.sol
│   │   ├── SafeCast.sol
│   │   ├── SqrtPriceMath.sol
│   │   ├── SwapMath.sol
│   │   ├── TickBitmap.sol
│   │   ├── TickMath.sol
│   │   ├── Tick.sol
│   │   ├── TransferHelper.sol
│   │   └── UnsafeMath.sol
```

```
│   ├── UniswapV3Factory.sol
│   ├── UniswapV3PoolDeployer.sol
│   ├── UniswapV3PoolImpl.sol
│   └── UniswapV3Pool.sol
├── periphery
│   ├── base
│   │   ├── BlockTimestamp.sol
│   │   ├── ERC721Permit.sol
│   │   ├── LiquidityManagement.sol
│   │   ├── Multicall.sol
│   │   ├── PeripheryImmutableState.sol
│   │   ├── PeripheryPayments.sol
│   │   ├── PeripheryPaymentsWithFee.sol
│   │   ├── PeripheryValidation.sol
│   │   ├── PoolInitializer.sol
│   │   └── SelfPermit.sol
│   ├── interfaces
│   │   ├── external
│   │   │   ├── IERC1271.sol
│   │   │   ├── IERC20PermitAllowed.sol
│   │   │   └── IWETH9.sol
│   │   ├── IERC20Metadata.sol
│   │   ├── IERC721Permit.sol
│   │   ├── IMulticall.sol
│   │   ├── INonfungiblePositionManager.sol
│   │   ├── INonfungibleTokenPositionDescriptor.sol
│   │   ├── IPeripheryImmutableState.sol
│   │   ├── IPeripheryPayments.sol
│   │   ├── IPeripheryPaymentsWithFee.sol
│   │   ├── IPoolInitializer.sol
│   │   ├── IQuoter.sol
│   │   ├── IQuoterV2.sol
│   │   ├── ISelfPermit.sol
│   │   ├── ISwapRouter.sol
│   │   ├── ITickLens.sol
│   │   └── IV3Migrator.sol
│   ├── lens
│   │   ├── Quoter.sol
│   │   ├── QuoterV2.sol
│   │   ├── README.md
│   │   ├── TickLens.sol
│   │   └── UniswapInterfaceMulticall.sol
│   ├── libraries
│   │   ├── BytesLib.sol
│   │   ├── CallbackValidation.sol
│   │   ├── ChainId.sol
│   │   ├── HexStrings.sol
│   │   ├── LiquidityAmounts.sol
│   │   ├── NFTDescriptor.sol
│   │   ├── NFTSVG.sol
│   │   ├── OracleLibrary.sol
```

```
│   │   ├── PoolAddress.sol
│   │   ├── PoolTicksCounter.sol
│   │   ├── PositionKey.sol
│   │   ├── PositionValue.sol
│   │   ├── SqrtPriceMathPartial.sol
│   │   ├── TokenRatioSortOrder.sol
│   │   └── TransferHelper.sol
│   ├── NonfungiblePositionManager.sol
│   ├── NonfungibleTokenPositionDescriptor.sol
│   ├── PositionMigrator.impl.sol
│   ├── PositionMigrator.sol
│   ├── SwapRouter.sol
│   ├── UniversalRouter.impl.sol
│   ├── UniversalRouter.sol
│   ├── V3AirdropOperator.sol
│   ├── V3Estimator.sol
│   ├── V3Migrator.sol
│   ├── V3Treasury.impl.sol
│   └── V3Treasury.sol
├── README.md
└── view
    ├── V3FactoryView.sol
    └── V3PositionView.sol
```

CHECK LIST

We reviewed source code based on the checklist below.

Basic bugs

- Access Control & Authorization
- Ownership Takeover
- Re-entrancy
- Integer Overflow/Underflow
- Wrong timestamp implementation
- DoS caused by wrong revert, infinite loop, etc..
- ERC20 idiosyncrasy

Business logic

- Properly implemented functionality
- Incorrect token/fee calculation
- Rounding errors
- Wrong implementation of feature
- Code asymmetries
- Synchronized state variables
- Governance token implementation

Specific scenario

Below are some scenarios that we have checked to find potential logic errors or vulnerabilities in V3 related contracts. '✓' checked scenarios indicate that there were no significant issues, and marked with '×' will be described in detail in the ***findings*** section.

- *NonfungiblePositionManager.sol*
  - ✓ Was the reward updated for positions done properly?
  - ✓ Was the reward calculation performed correctly?
  - × Is the modifier properly set?

- *V3Migrator.sol*

- ✓ Aren't tokens be frozen in the migrator contract during the migration process?
- ✓ Is it ensured that tokens are not migrated more than the v2 holding amount?
- ✓ Are the remaining tokens properly transferred?

- *PositionMigrator.impl.sol*
  - ✓ Is it possible for re-entrancy Attack through call-backs during **migrate** or **zap** function?
  - ✓ Isn't the refund amount exceeded?
  - ✗ Was the Fee calculation done correctly?
- *UniswapV3PoolImpl.sol*
  - ✓ Is the feeAmount appropriately distributed to buyback, pool voting, and fee growth?
  - ✓ Does uniswapCallback occur any re-entrancy vulnerabilities?
  - ✗ Was the Fee calculation done correctly?
- *V3Treasury.impl.sol*
  - ✓ Is the airdrop distributed within the totalAmount?
  - ✓ Is the calculation of the Amount based on block.number done correctly?
  - ✓ Is the distribution of the Amount ensured to be non-duplicative?
  - ✓ Was the initialization process performed correctly?
  - ✓ Are distributions id not duplicated?

FINDINGS

Impact classification

Severity	Description
High	This vulnerability affects a large number of users and has a critical impact on financial services.
Medium	This vulnerability affects the functionality of financial service but does not result in direct loss of funds.
Low	This vulnerability does not directly affect the service but if combined with other vulnerability can result in severe issue.

Summary

#	Title	Severity
1	Issue 1 - Unexpected behavior can happen during pool initialization	Medium
2	Issue2 - Wrong feeRatio calculation while swapping	High
3	Issue3 - Ineffective deadline check	Medium

Issue 1 - Incorrect compound fee handling

Summary	Severity
In *PositionMigrator.impl.sol*, **compoundFee** is used when user wants to migrate the position including the fees collected. However even if the fees were used to mint the position, it is still returned to user resulting in a potential loss of fund.	**Medium**

[Details]

migrate function in *PositionMigrator.impl.sol* is used to migrate users position in Uniswap V3 pool. The function works by first decreasing the liquidity and collecting the owed tokens.

```
// 1. Burn
{
    require(c.liquidity != 0, "Liquidity is 0");

    (uint256 burn0, uint256 burn1) = nonfungiblePositionManager.decreaseLiquidity(
        INonfungiblePositionManager.DecreaseLiquidityParams({
            tokenId: params.tokenId,
            liquidity : c.liquidity,
            amount0Min: params.burnAmount0Min,
            amount1Min: params.burnAmount1Min,
            deadline: block.timestamp
        })
    );

    require(burn0 >= params.burnAmount0Min, "Burn Slippage");
    require(burn1 >= params.burnAmount1Min, "Burn Slippage");

    (c.balance0, c.balance1, r.reward) = nonfungiblePositionManager.collect(
        INonfungiblePositionManager.CollectParams({
            tokenId: params.tokenId,
            recipient: address(this),
            amount0Max: type(uint128).max,
            amount1Max: type(uint128).max
        })
    );

    c.fee0 = c.balance0.sub(burn0);
    c.fee1 = c.balance1.sub(burn1);

    if (!params.compoundFee) {
        c.balance0 = c.balance0.sub(c.fee0);
        c.balance1 = c.balance1.sub(c.fee1);
    }
}
```

File : *PositionMigrator.impl.sol #199-231* **Function** : *migrate*

After ***collect*** function is called, **c.fee** is calculated by subtracting **c.balance** with **burn** so that if **compoundFee** is not enabled it can be subtracted from **c.balance**. In the end of the function, remaining tokens are refunded to user in a following way.

```
// refund tokens and transfer reward
token0.transfer(msg.sender, c.balance0.add(c.fee0).sub(r.amount0));
token1.transfer(msg.sender, c.balance1.add(c.fee1).sub(r.amount1));
IERC20Minimal(govToken).transfer(msg.sender, r.reward);
```

File : *PositionMigrator.impl.sol #292-294* **Function** : *migrate*

However code above always adds **c.fee** to **c.balance** regardless of **compoundFee** which means that even if user uses fees to mint position they still get the fee back resulting in loss of funds in **PositionMigrator** contract. User can even maximize this **c.fee** by calling **decreaseLiquidity** before calling **migrate** function.

Severity is **medium** as chances are unlikely that **PositionMigrator** contract contains any funds due to logic that refunds the remaining funds.

[Fix]

The code has been modified to correctly update the fee and balance when compoundFee is not set.

```
227 -           c.fee0 = c.balance0.sub(burn0);
228 -           c.fee1 = c.balance1.sub(burn1);
229
230             if (!params.compoundFee) {
231 -               c.balance0 = c.balance0.sub(c.fee0);
232 -               c.balance1 = c.balance1.sub(c.fee1);
```

```
229 +           nonfungiblePositionManager.burn(params.tokenId);
230
231             if (!params.compoundFee) {
232 +               c.fee0 = c.balance0.sub(burn0);
233 +               c.fee1 = c.balance1.sub(burn1);
234 +               c.balance0 = burn0;
235 +               c.balance1 = burn1;
```

Commit : 9df7aa5ba3c497ff9e55fdd4a51426bc999af40b

Issue 2 - Wrong feeRatio calculation while swapping

Summary	Severity
When swap is done between two tokens inside the pool, fee to be collected is calculated using **feeRatio**. This value is calculated incorrectly if pool is not boosted resulting in loss of fee.	High

[Details]

When **swap** function inside UniswapV3Impl.sol is called, **SwapState** structure is initialized before filling an order. **feeRatio** inside the structure is used for calculating the ratio of fee to liquidity providers. This variable is initialized by subtracting **feeShareRate** and **poolVotingRate** from 100.

```
SwapState memory state =
    SwapState({
        amountSpecifiedRemaining: amountSpecified,
        amountCalculated: 0,
        sqrtPriceX96: slot0Start.sqrtPriceX96,
        tick: slot0Start.tick,
        feeGrowthGlobalX128: zeroForOne ? feeGrowthGlobal0X128 : feeGrowthGlobal1X128,
        rewardGrowthGlobalX128: rewardGrowthGlobalX128,
        feeTotal: 0,
        feeRatio : uint256(100)
                    .sub(IGovernance(IUniswapV3Factory(factory).governance()).poolVotingRate())
                    .sub(IGovernance(IUniswapV3Factory(factory).governance()).feeShareRate()),
        liquidity: cache.liquidityStart
    });
```

File : *UniswapV3PoolImpl.sol #723-736* **Function** : *swap*

After order is filled **_updateFees** is called to collect the fees accrued. As shown in below if current pool doesn't consist of boosting tokens **poolVotingFee** becomes 0 and doesn't get collected.

```
function _updateFees(address token, uint feeAmount) private {
    IGovernance gov = IGovernance(IUniswapV3Factory(factory).governance());
    uint feeShareRate = gov.feeShareRate();
    address poolVoting = gov.poolVoting();
    address buyback = gov.buyback();
    uint buybackFee = feeAmount.mul(feeShareRate) / 100;
    uint poolVotingFee =
        (IPoolVoting(poolVoting).getPoolBoosting(address(this)) != 0) ?
        feeAmount.mul(gov.poolVotingRate()) / 100 : 0;

    if (buybackFee != 0) {
        TransferHelper.safeTransfer(token, buyback, buybackFee);
        (token == token0) ?
            IBuybackFund(buyback).updateFund0(buybackFee) :
            IBuybackFund(buyback).updateFund1(buybackFee);
    }

    if (poolVotingFee != 0) {
        TransferHelper.safeTransfer(token, poolVoting, poolVotingFee);
        (token == token0) ?
            IPoolVoting(poolVoting).marketUpdate0(poolVotingFee) :
            IPoolVoting(poolVoting).marketUpdate1(poolVotingFee);
    }
}
```

File : *UniswapV3PoolImpl.sol #885-908* **Function** : *_updateFees*

However, if you go back to the **feeRatio** calculation, **poolVotingRate** is subtracted no matter whether pool is boosted or not. This results in liquidity providers getting less fee then they should.

[Fix]

Implemented the ***getV3FeeRatio*** function to calculate the fee ratio correctly based on whether it is a mining pool or not.

```
733                 feeGrowthGlobalX128: zeroForOne ? feeGrowthGlobal0X128 : feeGrowthGlobal1X128,
734                 rewardGrowthGlobalX128: rewardGrowthGlobalX128,
735                 feeTotal: 0,
736 -               feeRatio : uint256(100)
737 -                           .sub(IGovernance(IUniswapV3Factory(factory).governance()).poolVotingRate())
738 -                           .sub(IGovernance(IUniswapV3Factory(factory).governance()).feeShareRate()),
739                 liquidity: cache.liquidityStart
740             });
741
```

```
734                 feeGrowthGlobalX128: zeroForOne ? feeGrowthGlobal0X128 : feeGrowthGlobal1X128,
735                 rewardGrowthGlobalX128: rewardGrowthGlobalX128,
736                 feeTotal: 0,
737 +               feeRatio :
        IGovernance(IUniswapV3Factory(factory).governance()).getV3FeeRatio(address(this)),
738                 liquidity: cache.liquidityStart
739             });
740
```

Commit : 1afa99735319ac5067f6d3bfeea89c4556bc2aa4

Issue 3 - Ineffective deadline check

Summary	Severity
When calling certain methods in **NonfungiblePositionManager** contract, **deadline** parameter is used to check whether transaction is still valid. However, this parameter is set to **block.timestamp** making the check ineffective.	Medium

[Details]

PositionMigrator has **migrate**, **zap** function which user can use to conveniently migrate their position in UniswapV3Pool. These functions interact with methods in **NonfungiblePositionManager** to manage positions.

```
(r.tokenId, r.liquidity, r.amount0, r.amount1) = nonfungiblePositionManager.mint(
    INonfungiblePositionManager.MintParams({
        token0: c.token0,
        token1: c.token1,
        fee: c.fee,
        tickLower: params.tickLower,
        tickUpper: params.tickUpper,
        amount0Desired: c.balance0,
        amount1Desired: c.balance1,
        amount0Min: params.mintAmount0Min,
        amount1Min: params.mintAmount1Min,
        recipient: msg.sender,
        deadline: block.timestamp
    })
);
```

File : *PositionMigrator.impl.sol #275-289* **Function** : *migrate*

When user has to increase/decrease liquidity or mint position, they have to pass the **deadline** parameter. This parameter is responsible for keeping transaction within the time user provide so that transaction doesn't get held back in the mempool forever. However as you can see above, this parameter is set to **block.timestamp** and the check is done as follows.

```
abstract contract PeripheryValidation is BlockTimestamp {
    modifier checkDeadline(uint256 deadline) {
        require(_blockTimestamp() <= deadline, 'Transaction too old');
        _;
    }
}
```

File : *PeripheryValidation.sol #6-11*

As **_blockTimestamp** returns **block.timestamp**, what happens here is contract is just comparing **block.timestamp** with **block.timestamp** which is always true making the deadline check useless.

Severity is **medium** as user can lose their funds up to maximum slippage amount, they provide or maybe more if the amount is not specified.

[Fix]

The code has been modified to accept the deadline as input from the user.

```
212                     liquidity : c.liquidity,
213                     amount0Min: params.burnAmount0Min,
214                     amount1Min: params.burnAmount1Min,
215 -                   deadline: block.timestamp
216                 })
217             );
218
```

```
214                     liquidity : c.liquidity,
215                     amount0Min: params.burnAmount0Min,
216                     amount1Min: params.burnAmount1Min,
217 +                   deadline: params.deadline
218                 })
219             );
220
```

Commit : f8cd78c255735359a85507e9749c81b6a2019c70

RECOMMENDATIONS

These are suggestions to improve code maintainability, readability, and/or resilience.

Summary

#	Title
1	Recommendation 1 - Unnecessary require statement

Recommendation 1 - Unnecessary require statement

zap function in *PositionMigrator.impl.sol* has require statement which checks whether swap slippage occurred.

```
        require(
            ISwapRouter(router).exactInputSingle(
                ISwapRouter.ExactInputSingleParams({
                    tokenIn: (params.zeroForOne) ? address(token0) : address(token1),
                    tokenOut: (params.zeroForOne) ? address(token1) : address(token0),
                    fee: fee,
                    recipient: address(this),
                    deadline: block.timestamp,
                    amountIn: amountIn,
                    amountOutMinimum: amountOut,
                    sqrtPriceLimitX96: 0
                })) >= amountOut,
            "Swap Slippage"
        );
    }
```

File : *PositionMigrator.impl.sol #326-340* **Function** : *zap*

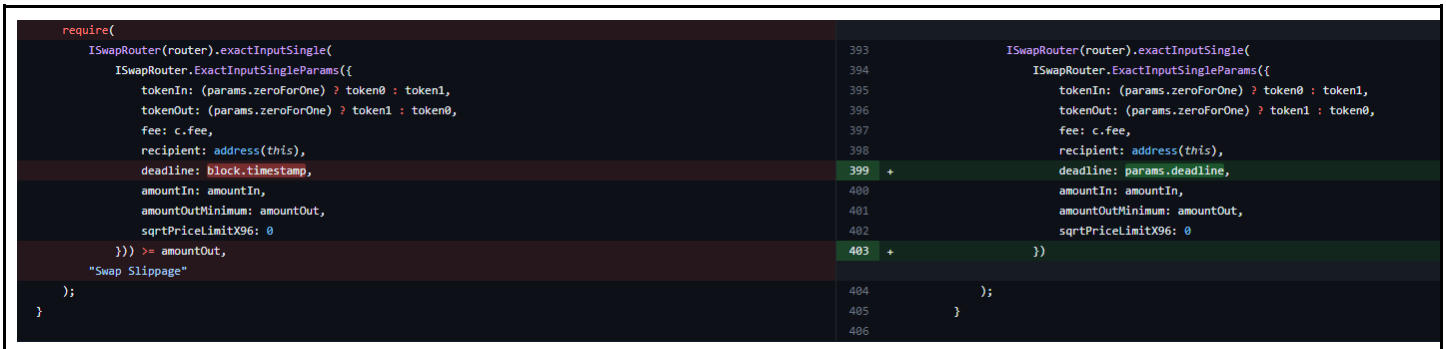
As slippage check is done inside the ***exactInputSingle*** function this require is unnecessary.

```
    /// @inheritdoc ISwapRouter
    function exactInputSingle(ExactInputSingleParams calldata params)
        external
        payable
        override
        checkDeadline(params.deadline)
        returns (uint256 amountOut)
    {
        amountOut = exactInputInternal(
            params.amountIn,
            params.recipient,
            params.sqrtPriceLimitX96,
            SwapCallbackData({path: abi.encodePacked(params.tokenIn, params.fee, params.tokenOut), payer: msg.sender})
        );
        require(amountOut >= params.amountOutMinimum, 'Too little received');
    }
```

File : *SwapRouter.sol #115-129* **Function** : *exactInputSingle*

[Fix]

Require statement has been removed.



```
    require(
        ISwapRouter(router).exactInputSingle(
            ISwapRouter.ExactInputSingleParams({
                tokenIn: (params.zeroForOne) ? token0 : token1,
                tokenOut: (params.zeroForOne) ? token1 : token0,
                fee: c.fee,
                recipient: address(this),
                deadline: block.timestamp,
                amountIn: amountIn,
                amountOutMinimum: amountOut,
                sqrtPriceLimitX96: 0
            })) >= amountOut,
        "Swap Slippage"
    );
}
```

```
                ISwapRouter(router).exactInputSingle(
                    ISwapRouter.ExactInputSingleParams({
                        tokenIn: (params.zeroForOne) ? token0 : token1,
                        tokenOut: (params.zeroForOne) ? token1 : token0,
                        fee: c.fee,
                        recipient: address(this),
+                       deadline: params.deadline,
                        amountIn: amountIn,
                        amountOutMinimum: amountOut,
                        sqrtPriceLimitX96: 0
+                   })

            );
        }

```

Commit : f8cd78c255735359a85507e9749c81b6a2019c70

APPENDIX

Slither

We used Slither as an automated analysis tool, and checked for dead code. But no significant issues were found, most of the issues were found in general libraries and OpenZeppelin. Below is the test environment and tool usage.

- **Repository** : https://github.com/crytic/slither.git

- **Test Environment** : Linux ubuntu-20 5.15.0-67-generic
  - How to install

```
sudo apt install python3-pip
python3 -m pip install slither-analyzer
pip3 install solc-select
solc-select install [version]
solc-select use [version]
```

- **Usage**
  - How to run

```
slither [target_contract_path] --solc-args=--optimize --solc-remaps
@openzeppelin=[openzeppelin_path] --checklist >> [output_file_name]
```

  - ✓ *--solc-args SOLC_ARGS* — Add custom solc arguments.
  - ✓ *--solc-remaps SOLC_REMAPS* — Add remapping.
  - ✓ *--checklist* — Generate a markdown page with the detector results.

- **Example**

```
- [ ] ID-177
[Address.functionStaticCall(address,bytes,string)](../../node_modules/@openzeppelin/contracts/utils/Address.so
45) is never used and should be removed

../../node_modules/@openzeppelin/contracts/utils/Address.sol#L139-L145


- [ ] ID-178
[Address.functionCall(address,bytes)](../../node_modules/@openzeppelin/contracts/utils/Address.sol#L79-L81) is
ed and should be removed

../../node_modules/@openzeppelin/contracts/utils/Address.sol#L79-L81


- [ ] ID-179
[AddressStringUtil.toAsciiString(address,uint256)](periphery/AddressStringUtil.sol#L7-L23) is never used and s
removed

periphery/AddressStringUtil.sol#L7-L23


- [ ] ID-180
[AddressStringUtil.char(uint8)](periphery/AddressStringUtil.sol#L28-L34) is never used and should be removed

periphery/AddressStringUtil.sol#L28-L34
```