QuillAudits

Audit Report

September, 2020

Contents

Introduction

This Audit Report highlights the overall security of GoldenGoose Smart Contract. With this report, we have tried to ensure the reliability of their smart contract by complete assessment of their system's architecture and the smart contract codebase.

Auditing Approach and Methodologies applied

The Quillhash team has performed thorough testing of the project starting with analysing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third party smart contracts and libraries.

Our team then performed a formal line by line inspection of the Smart Contract to find any potential issue like race conditions, transaction-ordering dependence, timestamp dependence, and denial of service attacks.

In the Unit testing Phase, we coded/conducted Custom unit tests written for each function in the contract to verify that each function works as expected. In Automated Testing, We tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was tested in collaboration of our multiple team members and this included -

- Testing the functionality of the Smart Contract to determine proper logic has been followed throughout the process.
- Analysing the complexity of the code by thorough, manual review of the code, line-by-line.
- Deploying the code on testnet using multiple clients to run live tests
- Analysing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
- Checking whether all the libraries used in the code are on the latest version.
- Analysing the security of the on-chain data.

Audit Details

Project Name: Momentum
Website/Etherscan Code (Mainnet): 0x9a7a4c141a3bcce4a31e42c1192ac6add35069b4
Website/Etherscan Code (Kovan): 0x696F93E6521a1097155790562E2f9ce33f445219
Languages: Solidity (Smart contract), Javascript (Unit Testing)
Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Securify, Mythril, Contract Library, Slither

The Contracts use 6 standard OpenZeppelin contract files, namely:
SafeMath.sol
Context.sol
Address.sol
IERC20.sol
Ownable.sol
ERC20.sol

The contract code was an exact match with OpenZeppelin's v3.1.0. And therefore can be labelled as well audited and tested contracts. Any non severe warning found during automated audit of these contracts will be of low concern.

Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient and working according to its specifications. The audit activities can be grouped in the following three categories:

Security
Identifying security related issues within each contract and the system of contracts.

Sound Architecture
Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.

Code Correctness and Quality
A full review of the contract source code. The primary areas of focus include:

- Correctness
- Readability
- Sections of code with high complexity
- Quantity and quality of test coverage

Security

Every issue in this report was assigned a severity level from the following:

High severity issues
They will bring problems and should be fixed.

Medium severity issues
They could potentially bring problems and should eventually be fixed.

Low severity issues
They are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Number of issues per severity

	Low	Medium	High
Open	8	5	0
Closed	0	0	0

Manual Audit

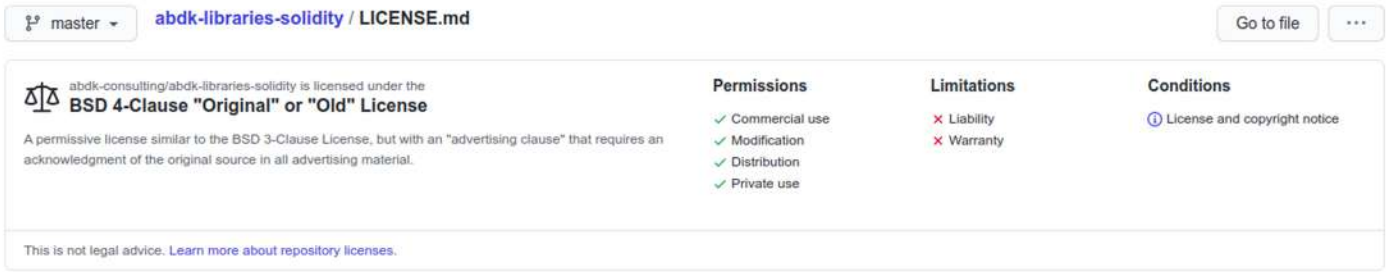
For this section the code was tested/read line by line by our developers. We also used Remix IDE's JavaScript VM and Kovan networks to test the contract functionality.

Low Level Severity Issues

1. The Momentum Contract uses ABDKMath64x64.sol library for signed 64 64-bit fixed point numbers. It has been recognized as a fixed point library in an OpenZepelin post. But they also point towards the weird licensing used by the library:

- `ABDKMath64x64` [2], by ABDK [3]. Smart contract library of mathematical functions operating with signed 64.64-bit fixed point numbers. ⚠ Weird license.

Github also recognizes the LICENSE as BSD 4-Clause "Original" or "Old" License.



Furthermore there were two closed issues on the github of the library stating concerns related to the Library's LICENSE by the community:

Issue1: https://github.com/abdk-consulting/abdk-libraries-solidity/issues/6
Issue2: https://github.com/abdk-consulting/abdk-libraries-solidity/issues/9

There is no suggested fix for this. Just a warning which should be explicit during the audit

2. Variable names are not very intuitive and therefore make the code hard to understand. Since, Solidity code is made public this concern cannot be ignored. Below, we will discuss some occurrences where this was noticed:

```
int128 SMweight;
int128 LMweightEx;
int128 LMweightCo;
```

Here SMweight should be replaced with shortMomentumWeight. Similarly LMweightEx and LMweightCo are not intuitive for readers to be differentiated as heavier or lighter from the variable names used.

Here _LMweightEx is of type uint256, which is being used to compute LMweightEx which is int128, the variable names should be updated to make this assignment more readable.

```
SMweight = Math.divu(_shortMomentumWeight, factor
LMweightEx = Math.divu(_LMweightEx, factor);
LMweightCo = Math.divu(_LMweightCo, factor);
```

Bad function name for getTransferFee. From the name of the function it looks like a simple getter function, but it is also used to update values of longMomentum and shortMomentum.

```
function getTransferFee(uint256 amount256, int128
    int128 amount = Math.fromUInt(amount256);
```

```
longMomentum = newLongMomentum;
shortMomentum = newShortMomentum;
```

3. Visibility of the variables mentioned in the below screenshot should be explicitly specified.

```
contract Momentum is ERC20, Ownable {
    int128 shortMomentum;
    int128 longMomentum;
    int128 SMweight;
    int128 LMweightEx;
    int128 LMweightCo;
```

4. VModularity of the destabilizing function can be improved. Similar calculations are performed twice in the same function:

> Math.mul(amount, Math.div(Math.fromUInt(1), Math.fromUInt(133)));

It is recommended to create an internal function for it.

```
function getDestabilizingTransferFee(int128 amount, int128 newRange, int128 newL
    // If newRange is within 2% of newLongMomentum, apply ~0.75% fee
    // to avoid potential overflow calculations and negative log values
    if (newRange <= Math.div(newLongMomentum, Math.fromUInt(50))) {
        return Math.mul(amount, Math.div(Math.fromUInt(1), Math.fromUInt(133)));
    } else {
        int128 proportion = Math.div(newRange, newLongMomentum);
        int128 rate = Math.add(Math.fromUInt(1), Math.ln(Math.mul(proportion, Ma
        return Math.mul(amount, Math.div(rate, Math.fromUInt(133)));
    }
}
```

5. Natspecs should be used to improve code readability.

6. Solidity integer division might truncate. As a result, performing multiplication before division might reduce precision.

7. The pragama versions used within these contracts are too recent and are not locked as well. Consider using version 0.5.11 for deploying the contracts. This change will affect contract code as well.

8. Individual contracts use different pragma versions. Different versions of Solidity is used in:

   Version used: ['>=0.6.12', '^0.5.0||^0.6.0', '^0.6.0', '^0.6.2']

   - ^0.6.0 (SafeMath.sol#3)
   - ^0.6.2 (Address.sol#3)
   - ^0.6.0 (IERC20.sol#3)
   - ^0.5.0||^0.6.0 (ABDKMath64x64.sol#5)
   - ^0.6.0 (Ownable.sol#3)
   - ^0.6.0 (Context.sol#3)
   - >=0.6.12 (Momentum.sol#2)
   - ^0.6.0 (ERC20.sol#3)

Medium Level Severity Issues

1. Unnecessary function overrides of allowance, increaseAllowance, decreaseAllowance and _approve function with no functionality addition was observed. All these functions are already defined within ERC20.sol.

```
function allowance(address owner, address spender) public view override re
    return _allowances[owner][spender];
}

function increaseAllowance(address spender, uint256 addedValue) public ove
    _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add
    return true;
}

function decreaseAllowance(address spender, uint256 subtractedValue) publi
    _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub
    return true;
}

function _approve(address owner, address spender, uint256 amount) internal
    require(owner != address(0), "ERC20: approve from the zero address");
    require(spender != address(0), "ERC20: approve to the zero address");

    _allowances[owner][spender] = amount;
    emit Approval(owner, spender, amount);
```

2. Function modifiers should be changed from public to external on multiple occurrences. Functions which are not called from within the contracts should be exposed as external functions to save gas during transaction execution.

```
function transfer(address recipient, uint256 amount256) public override returns
    burnAndTransfer(_msgSender(), recipient, amount256);
```

List of functions which should be external:

1. owner()
2. transferOwnership(address)
3. startBurning()
4. transfer(address,uint256)
5. transferFrom(address,address,uint256)
6. allowance(address,address)
7. increaseAllowance(address,uint256)

8. decreaseAllowance(address,uint256)
9. name()
10. symbol()
11. decimals()
12. balanceOf(address)
13. approve(address,uint256)

3. Bool isBurning should be a public variable. It would be helpful for the users to know if isburning has been enabled.

```
bool isBurning = false;
```

4. Momentum contract does not use custom events specific to the contract functionality. Events should be fired with all state variable updates as good practise. This makes it easier to build dApps on top of the contract's using existing tools.

5. Bad usage of Open Zeppelin's protocol:
Found duplicate declaration of variables within the Momentum contract. _allowances variable is already declared within ERC20.sol. Moreover, it was declared as a private variable and therefore was not intended to be used directly within contracts using openZeppelin's codebase.

```
mapping (address => mapping (address => uint256)) _allowances;
```

This is called state variable shadowing which is highly criticised.
Instead the allowances function should be used within overridden functions.

Wrong usage within Contract:

```
function transferFrom(address sender, address recipient, uint256 amount256) public override returns (bool) {
    burnAndTransfer(sender, recipient, amount256);
    _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount256, "ERC20: transfer amount exceeds allowance"));
    return true;
}
```

Right usage within Contract:

```
function transferFrom(address sender, address recipient, uint256 amount256) public override returns (bool) {
    burnAndTransfer(sender, recipient, amount256);
    _approve(sender, _msgSender(), allowance(sender, _msgSender()).sub(amount256, "ERC20: transfer amount exceeds allowance"));
    return true;
}
```

High Level Severity Issues

No high level severity issues

Automated Audit

Remix Compiler Warnings

It throws warnings by Solidity's compiler. If it encounters any errors the contract cannot be compiled and deployed.

```
browser/ABDKMath64x64.sol: Warning: SPDX license
identifier not provided in source file. Before
publishing, consider adding a comment containing
"SPDX-License-Identifier: <SPDX-License>" to each
source file. Use "SPDX-License-Identifier: UNLICENSED"
for non-open-source code. Please see https://spdx.org
for more information.
```

```
browser/ABDKMath64x64.sol:16:3: Warning: Documentation
tag on non-public state variables will be disallowed
in 0.7.0. You will need to use the @dev tag
explicitly. /** ^ (Relevant source part starts here
and spans across multiple lines).
```

```
browser/ABDKMath64x64.sol:21:3: Warning: Documentation
tag on non-public state variables will be disallowed
in 0.7.0. You will need to use the @dev tag
explicitly. /** ^ (Relevant source part starts here
and spans across multiple lines).
```

All the warnings correspond to the ABDKMath64x64 library.

Solhint Linting Violations

Solhint is an open-source project for linting solidity code, providing both security and style guide validations. It integrates seamlessly into most mainstream IDEs. We used Solhint as a plugin within our VScode for this analysis. Multiple Linting violations were detected by Solhint, we will cover an example of each violation in the section below:

Extra spaces found in import statements:

```
-import { ABDKMath64x64 as Math } from "./ABDKMath64x64.sol";
+import {ABDKMath64x64 as Math} from "./ABDKMath64x64.sol";
```

Extra spaces found in allowances declaration:

```
mapping (address => mapping (address => uint256)) _allowances;
mapping(address => mapping(address => uint256)) _allowances;
```

Extra space found near input arguments:

```
    uint256 initialSupply,
    uint256 initialSupply,
```

```
 uint256 _shortMomentum,
 uint256 _shortMomentum,
```

Maximum line length exceeded: Keeping lines under the PEP 8 recommendation to a maximum of 79 (or 99) characters helps readers easily parse the code.

```
function getMomentumAndSupply() external view returns (uint256, uint256, uint256) {
function getMomentumAndSupply()
    external
    view
    returns (
        uint256,
        uint256,
        uint256
    )
{
```

```
uint256 transferFee = getTransferFee(amount256, shortMomentum, longMomentum);
uint256 transferFee = getTransferFee(
    amount256,
    shortMomentum,
    longMomentum
);
```

Mythril

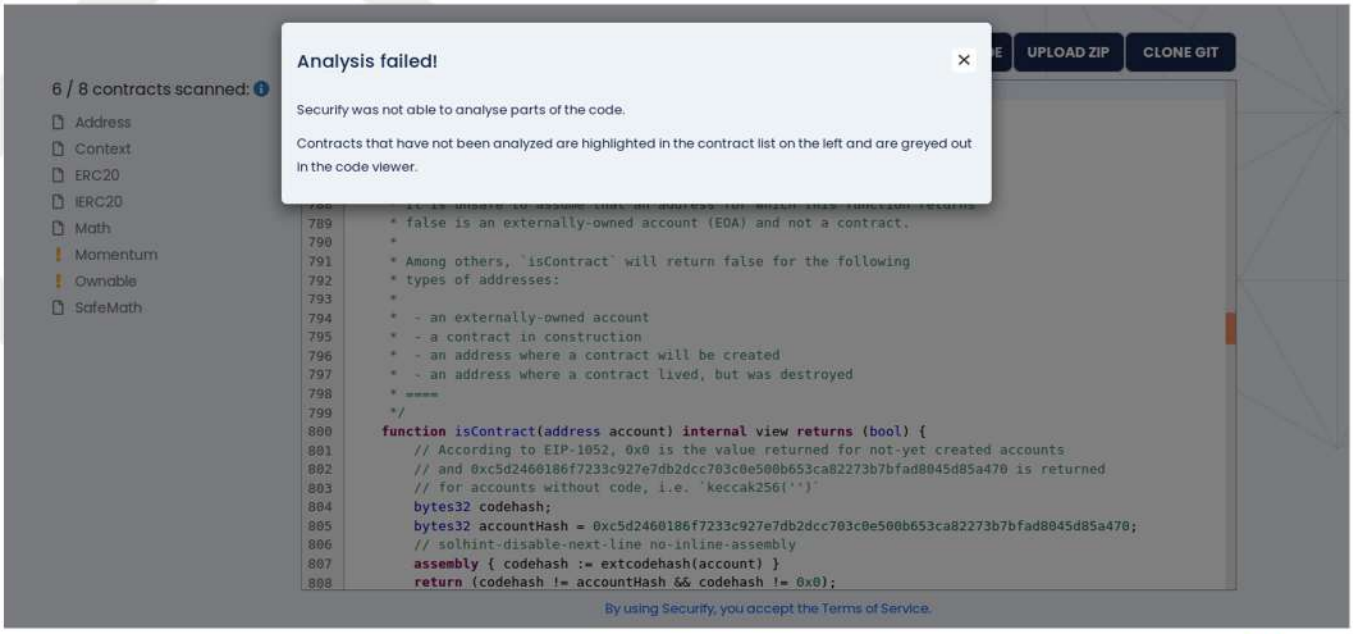
Mythril is a security analysis tool for EVM bytecode. It detects security vulnerabilities in smart contracts built for Ethereum, Hedera, Quorum, Vechain, Roostock, Tron and other EVM-compatible blockchains. It uses symbolic execution, SMT solving and taint analysis to detect a variety of security vulnerabilities. Mythril was used to analyse the contract code using runtime Bytecode of the contract. It indicated a possible integer underflow.

Note: this is a possible vulnerability usually thrown with all contracts containing complex mathematical calculations.

```
==== Integer Underflow ====
Type: Warning
Contract: MAIN
Function name: main
PC address: 54
A possible integer underflow exists in the function main.
The SUB instruction at address 54 may result in a value < 0.
--------------------
++++ Debugging info ++++
(5822) - (11748).]
```

Securify

Securify is a tool that scans Ethereum smart contracts for critical security vulnerabilities. Securify statically analyzes the EVM code of the smart contract to infer important semantic information (including control-flow and data-flow facts) about the contract. It was unable to Audit the Momentum contracts due to incompatible compiler versions.



Contract Library

Contract-library contains the most complete, high-level decompiled representation of all Ethereum smart contracts, with security analyses applied to these in real time.

We performed analysis using contract Library on the mainnet address of the Momentum contract: 0x9a7a4c141a3bcce4a31e42c1192ac6add35069b4

Analysis summary can be accessed here:
https://contract-library.com/contracts/Ethereum/0x9a7a4c141a3bcce4a31e42c1192ac6add35069b4

Also the data metrics of contracts deployed on mainnet can be accessed here: https://amberdata.io/addresses/0x9a7a4c141a3bcce4a31e42c1192ac6add35069b4/metrics

It did not return any issue during the analysis.

Slither

Slither, an open-source static analysis framework. This tool provides rich information about
Ethereum smart contracts and has the critical properties. While Slither is built as a security-oriented static analysis framework, it is also used to enhance the user's understanding of smart contracts, assist in code reviews, and detect missing optimizations.

All the notable issues of this analysis have already been considered above.

```
 momentum git:(audit) slither .
'npx truffle@5.1.39 compile --all' running (use --truffle-version truffle@x.x.x to use specific version)

Compiling your contracts...
===========================
> Compiling ./contracts/ABDKMath64x64.sol
> Compiling ./contracts/Address.sol
> Compiling ./contracts/Context.sol
> Compiling ./contracts/ERC20.sol
> Compiling ./contracts/IERC20.sol
> Compiling ./contracts/Momentum.sol
> Compiling ./contracts/Ownable.sol
> Compiling ./contracts/SafeMath.sol
> Compilation warnings encountered:
    /home/rails/work/audit/momentum/momentum/contracts/ABDKMath64x64.sol: Warning: SPDX license identifier not provided in source file. Before publishing, consider adding a comment containing "SPDX-License-Identifier: <SPDX-License>" to each source file. Use "SPDX-License-Identifier: UNLICENSED" for non-open-source code. Please see https://spdx.org for more information.
./home/rails/work/audit/momentum/momentum/contracts/ABDKMath64x64.sol:16:3: Warning: Documentation tag on non-public state variables will be disallowed in 0.7.0. You will need to use the @dev tag explicitly.
```

```
Address._functionCallWithValue(address,bytes,uint256,string) (Address.sol#119-140) uses assembly
   - INLINE ASM None (Address.sol#132-135)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Different versions of Solidity is used in :
   - Version used: ['>=0.6.12', '^0.5.0||^0.6.0', '^0.6.0', '^0.6.2']
   - ^0.6.0 (SafeMath.sol#3)
   - ^0.6.2 (Address.sol#3)
   - ^0.6.0 (IERC20.sol#3)
   - ^0.5.0||^0.6.0 (ABDKMath64x64.sol#5)
   - ^0.6.0 (Ownable.sol#3)
   - ^0.6.0 (Context.sol#3)
   - >=0.6.12 (Momentum.sol#2)
   - ^0.6.0 (ERC20.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
Pragma version^0.6.0 (SafeMath.sol#3) necessitates versions too recent to be trusted. Consider deploying with 0.5.11
Pragma version^0.6.2 (Address.sol#3) necessitates versions too recent to be trusted. Consider deploying with 0.5.11
Pragma version^0.6.0 (IERC20.sol#3) necessitates versions too recent to be trusted. Consider deploying with 0.5.11
Pragma version^0.5.0||^0.6.0 (ABDKMath64x64.sol#5) is too complex
Pragma version^0.6.0 (Ownable.sol#3) necessitates versions too recent to be trusted. Consider deploying with 0.5.11
Pragma version^0.6.0 (Context.sol#3) necessitates versions too recent to be trusted. Consider deploying with 0.5.11
Pragma version>=0.6.12 (Momentum.sol#2) necessitates versions too recent to be trusted. Consider deploying with 0.5.11
Pragma version^0.6.0 (ERC20.sol#3) necessitates versions too recent to be trusted. Consider deploying with 0.5.11
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (Address.sol#53-59):
   - (success) = recipient.call{value: amount}() (Address.sol#57)
Low level call in Address._functionCallWithValue(address,bytes,uint256,string) (Address.sol#119-140):
   - (success,returndata) = target.call{value: weiValue}(data) (Address.sol#123)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Function ABDKMath64x64.log_2(int256) (ABDKMath64x64.sol#366-389) is not in mixedCase
Function ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) is not in mixedCase
Constant ABDKMath64x64.MIN_64x64 (ABDKMath64x64.sol#19) is not in UPPER_CASE_WITH_UNDERSCORES
Constant ABDKMath64x64.MAX_64x64 (ABDKMath64x64.sol#24) is not in UPPER_CASE_WITH_UNDERSCORES
Parameter Momentum.getNewMomentum(int256,int256,int256).SMcopy (Momentum.sol#51) is not in mixedCase
Parameter Momentum.getNewMomentum(int256,int256,int256).LMcopy (Momentum.sol#51) is not in mixedCase
Parameter Momentum.getRangeData(int256,int256,int256,int256).LMcopy (Momentum.sol#58) is not in mixedCase
Parameter Momentum.getRangeData(int256,int256,int256,int256).SMcopy (Momentum.sol#58) is not in mixedCase
Parameter Momentum.getRangeData(int256,int256,int256,int256).LMnew (Momentum.sol#58) is not in mixedCase
Parameter Momentum.getRangeData(int256,int256,int256,int256).SMnew (Momentum.sol#58) is not in mixedCase
Parameter Momentum.getTransferFee(uint256,int256,int256).SMcopy (Momentum.sol#77) is not in mixedCase
Parameter Momentum.getTransferFee(uint256,int256,int256).LMcopy (Momentum.sol#77) is not in mixedCase
Variable Momentum._allowances (Momentum.sol#15) is not in mixedCase
Variable Momentum.SMweight (Momentum.sol#11) is not in mixedCase
Variable Momentum.LMweightEx (Momentum.sol#12) is not in mixedCase
Variable Momentum.LMweightCo (Momentum.sol#13) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
ABDKMath64x64.fromInt(int256) (ABDKMath64x64.sol#33-36) uses literals with too many digits:
   - require(bool)(x >= - 0x8000000000000000 && x <= 0x7FFFFFFFFFFFFFFF) (ABDKMath64x64.sol#34)
ABDKMath64x64.muli(int256,int256) (ABDKMath64x64.sol#144-170) uses literals with too many digits:
   - require(bool)(y >= - 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF && y <= 0x1000000000000000000000000000000000000000000000000
0000000) (ABDKMath64x64.sol#146-147)
ABDKMath64x64.muli(int256,int256) (ABDKMath64x64.sol#144-170) uses literals with too many digits:
   - require(bool)(absoluteResult <= 0x8000000000000000000000000000000000000000000000000000000000000000) (ABDKMath64x64.sol#161-162)
ABDKMath64x64.divi(int256,int256) (ABDKMath64x64.sol#219-239) uses literals with too many digits:
   - require(bool)(absoluteResult <= 0x80000000000000000000000000000000) (ABDKMath64x64.sol#233)
ABDKMath64x64.inv(int256) (ABDKMath64x64.sol#285-290) uses literals with too many digits:
   - result = int256(0x100000000000000000000000000000000) / x (ABDKMath64x64.sol#287)
ABDKMath64x64.gavg(int128,int128) (ABDKMath64x64.sol#311-317) uses literals with too many digits:
   - require(bool)(m < 0x4000000000000000000000000000000000000000000000000000000000000000) (ABDKMath64x64.sol#314-315)
ABDKMath64x64.pow(int256,uint256) (ABDKMath64x64.sol#327-347) uses literals with too many digits:
   - require(bool)(absoluteResult <= 0x8000000000000000000000000000000000) (ABDKMath64x64.sol#341)
ABDKMath64x64.sqrt(int128) (ABDKMath64x64.sol#355-358) uses literals with too many digits:
   - int128(sqrtu(uint256(x) << 64,0x10000000000000000)) (ABDKMath64x64.sol#357)
ABDKMath64x64.log_2(int256) (ABDKMath64x64.sol#366-389) uses literals with too many digits:
   - xc >= 0x10000000000000000 (ABDKMath64x64.sol#371)
ABDKMath64x64.log_2(int256) (ABDKMath64x64.sol#366-389) uses literals with too many digits:
   - xc >= 0x100000000 (ABDKMath64x64.sol#372)
```

```
ABDKMath64x64.log_2(int256) (ABDKMath64x64.sol#366-389) uses literals with too many digits:
    - bit = 0x8000000000000000 (ABDKMath64x64.sol#381)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - require(bool)(x < 0x400000000000000000) (ABDKMath64x64.sol#411)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x < - 0x400000000000000000 (ABDKMath64x64.sol#413)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = 0x80000000000000000000000000000000 (ABDKMath64x64.sol#415)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x8000000000000000 > 0 (ABDKMath64x64.sol#417)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x4000000000000000 > 0 (ABDKMath64x64.sol#419)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x2000000000000000 > 0 (ABDKMath64x64.sol#421)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x1000000000000000 > 0 (ABDKMath64x64.sol#423)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x800000000000000 > 0 (ABDKMath64x64.sol#425)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x400000000000000 > 0 (ABDKMath64x64.sol#427)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x200000000000000 > 0 (ABDKMath64x64.sol#429)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x100000000000000 > 0 (ABDKMath64x64.sol#431)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x80000000000000 > 0 (ABDKMath64x64.sol#433)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x40000000000000 > 0 (ABDKMath64x64.sol#435)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x20000000000000 > 0 (ABDKMath64x64.sol#437)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x10000000000000 > 0 (ABDKMath64x64.sol#439)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x8000000000000 > 0 (ABDKMath64x64.sol#441)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x4000000000000 > 0 (ABDKMath64x64.sol#443)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x2000000000000 > 0 (ABDKMath64x64.sol#445)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x1000000000000 > 0 (ABDKMath64x64.sol#447)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x800000000000 > 0 (ABDKMath64x64.sol#449)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x400000000000 > 0 (ABDKMath64x64.sol#451)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x200000000000 > 0 (ABDKMath64x64.sol#453)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x100000000000 > 0 (ABDKMath64x64.sol#455)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x100000B1721BCFC99D9F890EA06911763 >> 128 (ABDKMath64x64.sol#456)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x80000000000 > 0 (ABDKMath64x64.sol#457)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x10000058B90CF1E6D97F9CA14DBCC1628 >> 128 (ABDKMath64x64.sol#458)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x40000000000 > 0 (ABDKMath64x64.sol#459)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x1000002C5C863B73F016468F6BAC5CA2B >> 128 (ABDKMath64x64.sol#460)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x20000000000 > 0 (ABDKMath64x64.sol#461)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x100000162E430E5A18F6119E3C02282A5 >> 128 (ABDKMath64x64.sol#462)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x10000000000 > 0 (ABDKMath64x64.sol#463)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x1000000B1721835514B86E6D96EFD1BFE >> 128 (ABDKMath64x64.sol#464)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x8000000000 > 0 (ABDKMath64x64.sol#465)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x100000058B90C0B48C6BE5DF846C5B2EF >> 128 (ABDKMath64x64.sol#466)
```

```
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x4000000000 > 0 (ABDKMath64x64.sol#467)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x10000002C5C8601CC6B9E94213C72737A >> 128 (ABDKMath64x64.sol#468)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x2000000000 > 0 (ABDKMath64x64.sol#469)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x1000000162E42FFF037DF38AA2B219F06 >> 128 (ABDKMath64x64.sol#470)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x1000000000 > 0 (ABDKMath64x64.sol#471)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x10000000B17217FBA9C739AA5819F44F9 >> 128 (ABDKMath64x64.sol#472)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x800000000 > 0 (ABDKMath64x64.sol#473)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x1000000058B90BFCDEE5ACD3C1CEDC823 >> 128 (ABDKMath64x64.sol#474)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x400000000 > 0 (ABDKMath64x64.sol#475)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x100000002C5C85FE31F35A6A30DA1BE50 >> 128 (ABDKMath64x64.sol#476)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x200000000 > 0 (ABDKMath64x64.sol#477)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x10000000162E42FF0999CE3541B9FFFCF >> 128 (ABDKMath64x64.sol#478)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x100000000 > 0 (ABDKMath64x64.sol#479)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x100000000B17217F80F4EF5AADDA45554 >> 128 (ABDKMath64x64.sol#480)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x80000000 > 0 (ABDKMath64x64.sol#481)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x10000000058B90BFBF8479BD5A81B51AD >> 128 (ABDKMath64x64.sol#482)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x40000000 > 0 (ABDKMath64x64.sol#483)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x1000000002C5C85FDF84BD62AE30A74CC >> 128 (ABDKMath64x64.sol#484)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x20000000 > 0 (ABDKMath64x64.sol#485)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x100000000162E42FEFB2FED257559BDAA >> 128 (ABDKMath64x64.sol#486)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x10000000 > 0 (ABDKMath64x64.sol#487)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x1000000000B17217F7D5A7716BBA4A9AE >> 128 (ABDKMath64x64.sol#488)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x8000000 > 0 (ABDKMath64x64.sol#489)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x100000000058B90BFBE9DDBAC5E109CCE >> 128 (ABDKMath64x64.sol#490)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x4000000 > 0 (ABDKMath64x64.sol#491)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x10000000002C5C85FDF4B15DE6F17EB0D >> 128 (ABDKMath64x64.sol#492)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x2000000 > 0 (ABDKMath64x64.sol#493)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x1000000000162E42FEFA494F1478FDE05 >> 128 (ABDKMath64x64.sol#494)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x1000000 > 0 (ABDKMath64x64.sol#495)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x10000000000B17217F7D20CF927C8E94C >> 128 (ABDKMath64x64.sol#496)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x800000 > 0 (ABDKMath64x64.sol#497)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x1000000000058B90BFBE8F71CB4E4B33D >> 128 (ABDKMath64x64.sol#498)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x400000 > 0 (ABDKMath64x64.sol#499)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x100000000002C5C85FDF477B662B26945 >> 128 (ABDKMath64x64.sol#500)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x200000 > 0 (ABDKMath64x64.sol#501)
```

```
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x10000000000162E42FEFA3AE53369388C >> 128 (ABDKMath64x64.sol#502)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - x & 0x100000 > 0 (ABDKMath64x64.sol#503)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x100000000000B17217F7D1D351A389D40 >> 128 (ABDKMath64x64.sol#504)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x10000000000058B90BFBE8E8B2D3D4EDE >> 128 (ABDKMath64x64.sol#506)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x1000000000002C5C85FDF4741BEA6E77E >> 128 (ABDKMath64x64.sol#508)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x100000000000162E42FEFA39FE95583C2 >> 128 (ABDKMath64x64.sol#510)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x1000000000000B17217F7D1CFB72B45E1 >> 128 (ABDKMath64x64.sol#512)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x10000000000058B90BFBE8E7CC35C3F0 >> 128 (ABDKMath64x64.sol#514)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x1000000000002C5C85FDF473E242EA38 >> 128 (ABDKMath64x64.sol#516)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x100000000000162E42FEFA39F02B772C >> 128 (ABDKMath64x64.sol#518)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x10000000000000B17217F7D1CF7D83C1A >> 128 (ABDKMath64x64.sol#520)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x100000000000058B90BFBE8E7BDCBE2E >> 128 (ABDKMath64x64.sol#522)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x100000000000002C5C85FDF473DEA871F >> 128 (ABDKMath64x64.sol#524)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x1000000000000162E42FEFA39EF44D91 >> 128 (ABDKMath64x64.sol#526)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x10000000000000B17217F7D1CF79E949 >> 128 (ABDKMath64x64.sol#528)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x1000000000000058B90BFBE8E7BCE544 >> 128 (ABDKMath64x64.sol#530)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x1000000000000002C5C85FDF473DE6ECA >> 128 (ABDKMath64x64.sol#532)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x10000000000000162E42FEFA39EF366F >> 128 (ABDKMath64x64.sol#534)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x10000000000000B17217F7D1CF79AFA >> 128 (ABDKMath64x64.sol#536)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x1000000000000058B90BFBE8E7BCD6D >> 128 (ABDKMath64x64.sol#538)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x10000000000000002C5C85FDF473DE6B2 >> 128 (ABDKMath64x64.sol#540)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x10000000000000162E42FEFA39EF358 >> 128 (ABDKMath64x64.sol#542)
ABDKMath64x64.exp_2(int256) (ABDKMath64x64.sol#410-550) uses literals with too many digits:
    - result = result * 0x10000000000000B17217F7D1CF79AB >> 128 (ABDKMath64x64.sol#544)
ABDKMath64x64.exp(int128) (ABDKMath64x64.sol#558-565) uses literals with too many digits:
    - require(bool)(x < 0x400000000000000000) (ABDKMath64x64.sol#559)
ABDKMath64x64.exp(int128) (ABDKMath64x64.sol#558-565) uses literals with too many digits:
    - x < - 0x400000000000000000 (ABDKMath64x64.sol#561)
ABDKMath64x64.divuu(uint256,uint256) (ABDKMath64x64.sol#575-614) uses literals with too many digits:
    - xc >= 0x100000000 (ABDKMath64x64.sol#585)
ABDKMath64x64.powu(uint256,uint256) (ABDKMath64x64.sol#624-677) uses literals with too many digits:
    - 0x80000000000000000000000000000000 (ABDKMath64x64.sol#625)
ABDKMath64x64.powu(uint256,uint256) (ABDKMath64x64.sol#624-677) uses literals with too many digits:
    - xc >= 0x10000000000000000000000000000000 (ABDKMath64x64.sol#630)
ABDKMath64x64.powu(uint256,uint256) (ABDKMath64x64.sol#624-677) uses literals with too many digits:
    - xc >= 0x10000000000000000 (ABDKMath64x64.sol#631)
ABDKMath64x64.powu(uint256,uint256) (ABDKMath64x64.sol#624-677) uses literals with too many digits:
    - xc >= 0x100000000 (ABDKMath64x64.sol#632)
ABDKMath64x64.powu(uint256,uint256) (ABDKMath64x64.sol#624-677) uses literals with too many digits:
    - result = 0x80000000000000000000000000000000 (ABDKMath64x64.sol#643)
ABDKMath64x64.powu(uint256,uint256) (ABDKMath64x64.sol#624-677) uses literals with too many digits:
    - result >= 0x8000000000000000000000000000000000000000000000000000000000000000 (ABDKMath64x64.sol#651-652)
ABDKMath64x64.powu(uint256,uint256) (ABDKMath64x64.sol#624-677) uses literals with too many digits:
    - x >= 0x8000000000000000000000000000000000000000000000000000000000000000 (ABDKMath64x64.sol#662-663)
ABDKMath64x64.slitherConstructorConstantVariables() (ABDKMath64x64.sol#15-698) uses literals with too many digits:
    - MIN_64x64 = - 0x80000000000000000000000000000000 (ABDKMath64x64.sol#19)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
```

```
INFO:Detectors:
owner() should be declared external:
   - Ownable.owner() (Ownable.sol#35-37)
transferOwnership(address) should be declared external:
   - Ownable.transferOwnership(address) (Ownable.sol#63-67)
startBurning() should be declared external:
   - Momentum.startBurning() (Momentum.sol#109-112)
transfer(address,uint256) should be declared external:
   - ERC20.transfer(address,uint256) (ERC20.sol#117-120)
   - Momentum.transfer(address,uint256) (Momentum.sol#125-128)
transferFrom(address,address,uint256) should be declared external:
   - ERC20.transferFrom(address,address,uint256) (ERC20.sol#153-157)
   - Momentum.transferFrom(address,address,uint256) (Momentum.sol#130-134)
allowance(address,address) should be declared external:
   - Momentum.allowance(address,address) (Momentum.sol#136-138)
   - ERC20.allowance(address,address) (ERC20.sol#125-127)
increaseAllowance(address,uint256) should be declared external:
   - Momentum.increaseAllowance(address,uint256) (Momentum.sol#140-143)
   - ERC20.increaseAllowance(address,uint256) (ERC20.sol#171-174)
decreaseAllowance(address,uint256) should be declared external:
   - ERC20.decreaseAllowance(address,uint256) (ERC20.sol#190-193)
   - Momentum.decreaseAllowance(address,uint256) (Momentum.sol#145-148)
name() should be declared external:
   - ERC20.name() (ERC20.sol#66-68)
symbol() should be declared external:
   - ERC20.symbol() (ERC20.sol#74-76)
decimals() should be declared external:
   - ERC20.decimals() (ERC20.sol#91-93)
balanceOf(address) should be declared external:
   - ERC20.balanceOf(address) (ERC20.sol#105-107)
approve(address,uint256) should be declared external:
   - ERC20.approve(address,uint256) (ERC20.sol#136-139)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-as-external
INFO:Slither:. analyzed (8 contracts with 46 detectors), 166 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Unit Tests

We used open zeppelin's test-helpers and test-environment to write these contract test cases and executed them using a combination of truffle, mocha and chai.

```
   momentum git:(audit) npm test

> @ test /home/rails/work/audit/momentum/momentum
> npx mocha --exit --recursive test --timeout 12000



  Momentum
    has a name
    has a symbol
    has 10 decimals
    has long momentum
    has short momentum
    has an owner (43ms)
    total supply
      returns the total amount of tokens
    balanceOf
      when the requested account has no tokens
        returns zero
      when the requested account has some tokens
        returns the total amount of tokens
    transfer
      when the recipient is not the zero address
```

```
    when the sender does not have enough balance
      □ reverts (102ms)
    when the sender transfers all balance
      □ transfers the requested amount (68ms)
      □ emits a transfer event (38ms)
    when the sender transfers zero tokens
      □ transfers the requested amount (83ms)
      □ emits a transfer event
  when the recipient is the zero address
    □ reverts
transfer from
  when the token owner is not the zero address
    when the recipient is not the zero address
      when the spender has enough approved balance
        when the token owner has enough balance
          □ transfers the requested amount (59ms)
          □ decreases the spender allowance (46ms)
          □ emits a transfer event
          □ emits an approval event (40ms)
        when the token owner does not have enough balance
          □ reverts (40ms)
      when the spender does not have enough approved balance
        when the token owner has enough balance
          □ reverts (38ms)
        when the token owner does not have enough balance
          □ reverts
    when the recipient is the zero address
      □ reverts
  when the token owner is the zero address
    □ reverts
approve
  when the spender is not the zero address
    when the sender has enough balance
      □ emits an approval event
      when there was no approved amount before
        □ approves the requested amount (40ms)
      when the spender had an approved amount
        □ approves the requested amount and replaces the previous one
    when the sender does not have enough balance
      □ emits an approval event
      when there was no approved amount before
        □ approves the requested amount (42ms)
      when the spender had an approved amount
        □ approves the requested amount and replaces the previous one (52ms)
  when the spender is the zero address
    □ reverts
get momentum and supply
  □ returns supply
  □ returns initial long momentum
  □ returns initial short momentum
decrease allowance
  when the spender is not the zero address
    when the sender has enough balance
      when there was no approved amount before
        □ reverts
      when the spender had an approved amount
        □ emits an approval event
        □ decreases the spender allowance subtracting the requested amount (39ms)
        □ sets the allowance to zero when all allowance is removed
        □ reverts when more than the full allowance is removed
    when the sender does not have enough balance
      when there was no approved amount before
        □ reverts
      when the spender had an approved amount
        □ emits an approval event
        □ decreases the spender allowance subtracting the requested amount (50ms)
        □ sets the allowance to zero when all allowance is removed
        □ reverts when more than the full allowance is removed
  when the spender is the zero address
    □ reverts
increase allowance
```

```
  when the spender is not the zero address
    when the sender has enough balance
      ✓ emits an approval event
      when there was no approved amount before
        ✓ approves the requested amount (47ms)
      when the spender had an approved amount
        ✓ increases the spender allowance adding the requested amount
    when the sender does not have enough balance
      ✓ emits an approval event
      when there was no approved amount before
        ✓ approves the requested amount (50ms)
      when the spender had an approved amount
        ✓ increases the spender allowance adding the requested amount (54ms)
  when the spender is the zero address
    ✓ reverts
start burning
  ✓ reverts when not called by owner
  ✓ renounces ownership
  transfer
    burns transfer fee for changing momentum
      ✓ decrements totalSupply
      ✓ decrements recipient balance
      ✓ emits Transfer event
      ✓ does not change momentum if amount is less than ~0.3% of lower momentum value
      ✓ changes momentum for Negative delta (130ms)
      ✓ changes momentum for Positive delta (LMweigthEx) (71ms)
      ✓ changes momentum for Positive delta (LMweigthCo) (74ms)
  transferFrom
    burns transfer fee for changing momentum
      ✓ decrements totalSupply
      ✓ decrements recipient balance
      ✓ emits Transfer event
      ✓ does not change momentum if amount is less than ~0.3% of lower momentum value
      ✓ changes momentum for Negative delta (150ms)
      ✓ changes momentum for Positive delta (LMweigthEx) (67ms)
      ✓ changes momentum for Positive delta (LMweigthCo) (81ms)

68 passing (9s)
```

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the Momentum contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Summary

Use case of the smart contract is very well designed and Implemented. Altogether, the code is written and demonstrates effective use of abstraction, separation of concerns, and modularity. But there are a number of issues/vulnerabilities to be tackled in the medium level severity and low level severity, code is readable but can be improved according to the Solidity's style guide which is recommended to be fixed before implementing a live version.

QuillAudits

448-A EnKay Square, Opposite Cyber Hub,
Gurugram, Harayana, India - 122016

audits.quillhash.com

hello@quillhash.com