HAECHI AUDIT

Mspace

Smart Contract Security Analysis

Published on : Dec 15, 2021

Version v1.0

HAECHI AUDIT

Smart Contract Audit Certificate

Mspace

Security Report Published by HAECHI AUDIT
v1.0 Dec 15, 2021

Auditor : Jasper Lee

Executive Summary

Severity of Issues	Findings	Resolved	Unresolved	Acknowledged	Comment
Critical	-	-	-	-	-
Major	-	-	-	-	-
Minor	-	-	-	-	-
Tips	4	-	-	-	-

TABLE OF CONTENTS

4 Issues (0 Critical, 0 Major, 0 Minor, 4 Tips) Found

ABOUT US

HAECHI AUDIT believes in the power of cryptocurrency and the next paradigm it will bring. We have the vision to empower the next generation of finance. By providing security and trust in the blockchain industry, we dream of a world where everyone has easy access to blockchain technology.

HAECHI AUDIT is a flagship service of HAECHI LABS, the leader of the global blockchain industry. HAECHI AUDIT provides specialized and professional smart contract security auditing and development services.

We are a team of experts with years of experience in the blockchain field and have been trusted by 300+ project groups. Our notable partners include Sushiswap,1inch, Klaytn, Badger, etc.

HAECHI AUDIT is the only blockchain technology company selected for the Samsung Electronics Startup Incubation Program in recognition of our expertise. We have also received technology grants from the Ethereum Foundation and Ethereum Community Fund.

Inquiries: audit@haechi.io
Website: audit.haechi.io

INTRODUCTION

This report was prepared to audit the security of Mspace smart contract created by Mspace project team. HAECHI AUDIT conducted the audit focusing on whether the smart contract created by Mspace project team is soundly implemented and designed as specified in the published materials, in addition to the safety and security of the smart contract.

CRITICAL	Critical issues must be resolved as critical flaws that can harm a wide range of users.
MAJOR	Major issues require correction because they either have security problems or are implemented not as intended.
MINOR	Minor issues can potentially cause problems and therefore require correction.
TIPS	Tips issues can improve the code usability or efficiency when corrected.

HAECHI AUDIT recommends the Mspace project team improve all issues discovered.

The following issue explanation uses the format of {file name}#{line number}, {contract name}#{function/variable name} to specify the code. For instance, Sample.sol:20 points to the 20th line of Sample.sol file, and Sample#fallback() means the fallback() function of the Sample contract.

Please refer to the Appendix to check all results of the tests conducted for this report.

SUMMARY

The codes used in this Audit can be found at GitHub (https://gitlab.com/mspace.ai/mspace-swap). The last commit of the code used for this Audit is affa1ee7c5b7eafec47f92f05955f3b3d285e69b.

Issues HAECHI AUDIT found 0 critical issues, 0 major issues, and 0 minor issues. There are 4 Tips issues explained that would improve the code's usability or efficiency upon modification.

Severity	Issue	Status
TIPS	Meaningless guard statement in *BoringHelper#getMETARate()*	(Found - v1.0)
TIPS	*MiningHill#emergencyWithdraw()* does not follow the checks-effects-interactions	(Found - v1.0)
TIPS	Inefficient storage usage in *UniswapV2Factory*	(Found - v1.0)
TIPS	Possible sandwich/MEV attacks	(Found - v1.0)

OVERVIEW

Contracts subject to audit

- Timelock
- WMETA
- ENS
- IERC20
- IFactory
- IMiningHill
- IOracle
- IPair
- IStrategy
- BoringERC20
- BoringMath
- BoringPair
- IOwned
- Owned
- Utils
- BancorConverterRegistry
- BoringHelper
- ENSRegistry
- ENSRegistryWithFallback
- Multicall2
- UpgradeableProxy
- IMSpace
- SafeERC20
- SafeMath
- EternalStorage
- MiningHill
- MiningPoolMigrator
- Extractor
- MSpaceMintPlan
- MSpaceToken
- Bank
- IERC20Uniswap
- IUniswapV2Callee
- IUniswapV2ERC20
- IUniswapV2Factory

- IUniswapV2Pair
- IUniswapV2Router01
- IUniswapV2Router02
- IWMETA
- Math
- SafeMathUniswap
- TransferHelper
- UniswapV2Library
- UQ112x112
- UniswapV2ERC20
- UniswapV2Factory
- UniswapV2Migrator
- UniswapV2Pair
- UniswapV2Router02

Mspace smart contract has the following privileges.

- Admin
- Owner
- Team

Each privilege can access the following functions.

Role	Functions
Owner	*BoringHelper#setContracts()* *MiningHill#add()* *MiningHill#set()* *MiningHill#setMigrator()* *Extractor#setBridge()* *MSpaceMintPlan#setStrategy()* *MSpaceMintPlan#setStrategyPoint()* *MSpaceMintPlan#setBlockStep()* *MSpaceToken#mint()* *UniswapV2Factory#setPairImplement()* *UniswapV2Factory#lockPair()* *UniswapV2Factory#unlockPair()* *UniswapV2Factory#setFeeTo()* *UniswapV2Factory#setMigrator()* *UniswapV2Pair#setLock()*
Team	*MSpaceToken#setTeam()*
Admin	*TimeLock#queueTransaction()* *TimeLock#cancelTransaction()* *TimeLock#executeTransaction()*

FINDINGS

💡 **TIPS**

Meaningless guard statement in *BoringHelper#getMETARate()*

(Found - v.1.0)

```
function getMETARate(IERC20 token) public view returns (uint256) {
    if (token == WMETA) {
        return 1e18;
    }
    IPair pair;

    if (factory != IFactory(address(0))) {
        pair = IPair(factory.getPair(token, WMETA));
    }
    if (address(pair) == address(0)) {
        return 0;
    }

    uint112 reserve0;
    uint112 reserve1;
    IERC20 token0;

    if (address(pair) != address(0)) {
        (uint112 reserve0MSpace, uint112 reserve1MSpace, ) = pair.getReserves();
        reserve0 += reserve0MSpace;
        reserve1 += reserve1MSpace;
        if (token0 == IERC20(address(0))) {
            token0 = pair.token0();
        }
    }
```

[https://gitlab.com/mspace.ai/mspace-swap/-/blob/affa1ee7c5b7eafec47f92f05955f3b3d285e69b/contracts/helpers/BoringHelper.sol#L63-L87]

Issue

token0 is declared in L78 and must be a zero address. Therefore, the guard statement to check the zero address of L84 is meaningless.

Recommendation

Please remove the guard statement in Line 84.

TIPS

MiningHill#emergencyWithdraw() does not follow the checks-effects-interactions (Found - v.1.0)

```
function emergencyWithdraw(uint256 pid) public {
  updatePool(pid);
  updateUser(pid, msg.sender);
  PoolInfo storage pool = poolInfo[pid];
  UserInfo storage user = userInfo[pid][msg.sender];
  pool.lpToken.safeTransfer(msg.sender, user.amount);
  user.amount = 0;
  emit EmergencyWithdraw(msg.sender, pid, user.amount);
}
```

[https://gitlab.com/mspace.ai/mspace-swap/-/blob/affa1ee7c5b7eafec47f92f05955f3b3d285e69b/contracts/stake/MiningHill.sol#L234-L242]

Issue

The value *user.amount* is not zeroed until after the external call to *pool.lpToken.safeTransfer()*. Although the LP tokens are assumed trusted as of now, if a compromised token is added, reentrancy may allow the theft of tokens.

Recommendation

In general we recommend following the checks-effects-interactions pattern. Zero out the user's state variables prior to the external *safeTransfer()* call.

TIPS

Inefficient storage usage in *UniswapV2Factory*

(Found - v.1.0)

```
    function createPair(address tokenA, address tokenB) external override returns (address pair) {
        require(tokenA != tokenB, 'UniswapV2: IDENTICAL_ADDRESSES');
        (address token0, address token1) = tokenA < tokenB ? (tokenA, tokenB) : (tokenB, tokenA);
        require(token0 != address(0), 'UniswapV2: ZERO_ADDRESS');
        require(getPair[token0][token1] == address(0), 'UniswapV2: PAIR_EXISTS'); // single check is
sufficient
        // bytes memory bytecode = type(UniswapV2Pair).creationCode;
        // bytes32 salt = keccak256(abi.encodePacked(token0, token1));
        // assembly {
        //     pair := create2(0, add(bytecode, 32), mload(bytecode), salt)
        // }
        UpgradeableProxy pairProxy = new UpgradeableProxy(
          address(pairImplement), address(this), bytes("")
        );
        pair = address(pairProxy);
        IUniswapV2Pair(address(pair)).initialize(token0, token1);
        getPair[token0][token1] = pair;
        getPair[token1][token0] = pair; // populate mapping in the reverse direction
        allPairs.push(pair);
        emit PairCreated(token0, token1, pair, allPairs.length);
    }
```

[https://gitlab.com/mspace.ai/mspace-swap/-/blob/affa1ee7c5b7eafec47f92f05955f3b3d285e69b/contracts/uniswapv2/UniswapV2Factory.sol#L60-L79]

Issue

token0 and *token1* are sorted values. Therefore, when two token addresses are given, we can distinguish which token is *token0*. Mapping *getPair* with reverse direction is an inefficient use of storage because it costs gas more than finding a pair by comparing the two addresses.

Recommendation

Please remove reversed mapping and fix logic finding pair address.

💡 **TIPS**

Possible sandwich/MEV attack

(Found - v.1.0)

Issue

A large-scale transaction can be sandwiched by prior selling to lower market prices, and there can be a tailgate buyback that adds the same amount to the transaction amount. This sandwich attack can cause losses and brings less profit to trading users as expected.

Recommendation

Develop a measure to the above front-running attack.

DISCLAIMER

This report does not guarantee investment advice, the suitability of the business models, and codes that are secure without bugs. This report shall only be used to discuss known technical issues. Other than the issues described in this report, undiscovered issues may exist such as defects in Ethereum and Solidity. To write secure smart contracts, correction of discovered problems and sufficient testing thereof are required.

Appendix A. Test Results

The following results are unit test results that cover the key logic of the smart contract subject to the security audit. Parts marked in red are test cases that failed to pass the test due to having issues.

```
Timelock
  #constructor()
    ✓ Should fail if delay is smaller than minimum delay
    ✓ Should fail if delay exceed maximum delay
    ✓ Should fail if already initalized
    ✓ Should initialize timelock parameters
  #setDelay()
    ✓ Should fail if caller is not timelock
    ✓ Should fail if delay is smaller than minimum delay
    ✓ Should fail if delay exceed maximum delay
    ✓ Should update delay time
    ✓ Should emit NewDelay event
  #setPendingAdmin()
    ✓ Should fail if caller is not timelock when admin is initalized
    ✓ Should fail if caller is not admin when admin is not initalized
    ✓ Should update pending admin
    ✓ Should emit NewPendingAdmin event
  #acceptAdmin()
    ✓ Should fail if caller is not pending admin
    ✓ Should update admin
    ✓ Should emit NewAdmin event
  #queueTransaction()
    ✓ Should fail if caller is not admin
    ✓ Should fail if expected execution block is not satisfy delay
    ✓ Should add queued transaction list
    ✓ Should emit QueueTransaction event
  #cancelTransaction()
    ✓ Should fail if caller is not admin
    ✓ Should update queued transaction list
    ✓ Should emit CancelTransaction event
  #executeTransaction()
    ✓ Should fail if caller is not admin
    ✓ Should fail if there is no queued transaction
    ✓ Should fail if the transaction doesn't surpassed timelock
    ✓ Should fail if the transaction is stale
    ✓ Should fail if the transaction reverted
    ✓ Should return returnData
    ✓ Should emit ExecuteTransaction event
  #getBlockTimestamp()
    ✓ Should return block timestamp
```

```
WMETA
  #deposit()
    ✓ Should update user balance
    ✓ Should emit Deposit event
  #withdraw()
    ✓ Should fail if withdrawal amount is larger than balance
    ✓ Should update user balance
    ✓ Should withdraw meta
    ✓ Should emit Withdrawal event
  #totalSupply()
    ✓ Should return total supply
  #approve()
    ✓ Should update user allowance
    ✓ Should emit Approval event
  #transferFrom()
    ✓ Should fail if transfer amount is larger than balance
    ✓ Should fail if transfer amount is larger than allowance
    ✓ Should update allowance
    ✓ Should update users balance
    ✓ Should emit Transfer event
  #transfer()
    ✓ Should fail if transfer amount is larger than balance
    ✓ Should update users balance
    ✓ Should emit Transfer event

BancorConverterRegistry
  #tokenCount()
    ✓ Should return the number of tokens in the registry
  #converterCount()
    ✓ Should return the number of converters associated with the given token
  #converterAddress()
    ✓ Should return the converter address
  #tokenAddress()
    ✓ Should return the token address
  #registerConverter()
    ✓ Should fail if the caller is not owner
    ✓ Should fail if the token address is zero address
    ✓ Should fail if the converter address is zero address
    ✓ Should fail if the converter address and token address is already associated
    ✓ Should update converter params
    ✓ Should emit ConverterAddition event
  #unregisterConverter()
    ✓ Should fail if the caller is not owner
    ✓ Should fail if the token address is zero address
    ✓ Should fail if the converter index exceed tokensToConverters list length
    ✓ Should remove converter params
    ✓ Should emit ConverterRemoval event
```

```
BoringHelper
  #constructor()
    ✓ Should initalize contract params
  #setContracts()
    ✓ Should fail if caller is not owner
    ✓ Should update contract params
  #getMETARate()
    ✓ Should return 1e18 if token is WMETA
    ✓ Should return 0 if pair not exists
    ✓ Should return meta price if pair exists
  #getUIInfo()
    ✓ Should return info for user interface (currency is zero address)
    ✓ Should return info for user interface (WETH is zero address)
    ✓ Should return info for user interface (mspace is zero address)
    ✓ Should return info for user interface (space is zero address)
    ✓ Should return info for user interface (gravity is zero address)
    ✓ Should return info for user interface
  #getTokenInfo()
    ✓ Should return token info
  #findBalances()
    ✓ Should return simple info of token balances
  #getBalances()
    ✓ Should return full info of token balances
  #getPairs()
    ✓ Should return pair info
  #pollPairs()
    ✓ Should return pair user info
  #getPools()
    ✓ Should return pool info
  #findPools()
    ✓ Should return pool that matches pid
  #pollPools()
    ✓ Should return pool user info

ENSRegistry
  #constructor()
    ✓ Should initalize owner
  #setOwner()
    ✓ Should fail if caller is not node owner
    ✓ Should update owner of node
    ✓ Should emit Transfer event
  #setRecord()
    ✓ Should fail if caller is not node owner
    ✓ Should update resolver
    ✓ Should emit NewResolver event
    ✓ Should update ttl
    ✓ Should emit NewTTL event
  #setSubnodeOwner()
    ✓ Should fail if caller is not node owner
```

```
    ✓ Should update owner
    ✓ Should emit NewOwner
  #setSubnodeRecord()
    ✓ Should fail if caller is not node owner
    ✓ Should update resolver
    ✓ Should emit NewResolver event
    ✓ Should update ttl
    ✓ Should emit NewTTL event
  #setResolver()
    ✓ Should fail if caller is not node owner
    ✓ Should update resolver
    ✓ Should emit NewResolver event
  #setTTL()
    ✓ Should fail if caller is not node owner
    ✓ Should update ttl
    ✓ Should emit NewTTL event
  #setApprovalForAll()
    ✓ Should enable operator for all
    ✓ Should emit ApprovalForAll event
  #owner()
    ✓ Should return owner for record
  #resolver()
    ✓ Should return resolver for record
  #ttl()
    ✓ Should return ttl for record
  #recordExists()
    ✓ Should return true if record exists
  #isApprovedForAll()
    ✓ Should return true if owner approved an operator

ENSRegistryWithFallback
  #constructor()
    ✓ Should initalize old ENS contract
  #resolver()
    ✓ Should return resolver for record (find at old ENS)
    ✓ Should return resolver for record
  #owner()
    ✓ Should return owner for record (find at old ENS)
    ✓ Should return owner for record
  #ttl()
    ✓ Should return ttl for record (find at old ENS)
    ✓ Should return ttl for record
  #setOwner()
    ✓ Should fail if caller is not node owner
    ✓ Should update owner of node
    ✓ Should update owner to ENS contract address if _owner is zero address
    ✓ Should emit Transfer event

Multicall2
```

```
  #aggregate()
    ✓ Should call multiple calls
    ✓ Should fail if a call fails
  #blockAndAggregate()
    ✓ Should call multiple calls
    ✓ Should return block info
  #getBlockHash()
    ✓ Should return block hash
  #getBlockNumber()
    ✓ Should return block number
  #getCurrentBlockCoinbase()
    ✓ Should return block coinbase
  #getCurrentBlockDifficulty()
    ✓ Should return block difficulty
  #getCurrentBlockGasLimit()
    ✓ Should return block gas limit
  #getCurrentBlockTimestamp()
    ✓ Should return block timestamp
  #getMetaBalance()
    ✓ Should return meta balance of address
  #getLastBlockHash()
    ✓ Should return last block hash
  #tryAggregate()
    ✓ Should call multiple calls
    ✓ Should fail if a call fails if required
  #tryBlockAndAggregate()
    ✓ Should call multiple calls
    ✓ Should return block info

UpgradeableProxy
  #constructor()
    ✓ Should fail if admin address is zero address
    ✓ Should initalize contract logic and data, admin
  #admin()
    ✓ Should return admin address
  #implementation()
    ✓ Should return implementation address
  #changeAdmin()
    ✓ Should fail if caller is not admin
    ✓ Should update admin
    ✓ Should emit AdminChanged event
  #upgradeTo()
    ✓ Should fail if caller is not admin
    ✓ Should upgrade implementation
  #upgradeToAndCall()
    ✓ Should fail if caller is not admin
    ✓ Should upgrade implementation
    ✓ Should call a function after upgrade
```

```
EternalStorage
  #setAddress()
    ✓ Should set address in storage
  #getAddress()
    ✓ Should get address in storage
  #setUint()
    ✓ Should set uint in storage
  #getUint()
    ✓ Should get uint in storage
  #setString()
    ✓ Should set string in storage
  #getString()
    ✓ Should get string in storage
  #setBool()
    ✓ Should set bool in storage
  #getbool()
    ✓ Should get bool in storage
  #setBytes()
    ✓ Should set bytes in storage
  #getBytes()
    ✓ Should get bytes in storage
  #setInt()
    ✓ Should set int in storage
  #getInt()
    ✓ Should get int in storage

MiningHill
  #init()
    ✓ Should fail if startRewrd is before current block number
    ✓ Should fail if blockstep is zero
    ✓ Should fail if msp address is zero address
    ✓ Should fail if treasury address is zero address
    ✓ Should fail if already initalized
    ✓ Should initalize contract params
  #poolLength()
    ✓ Should return pool length
  #add()
    ✓ Should fail if caller is not owner
    ✓ Should fail if pool already exists
    ✓ Should add pool info (withUpdate)
    ✓ Should add pool info
    ✓ Should emit AddPool event
  #set()
    ✓ Should fail if caller is not owner
    ✓ Should fail if pool doesn't exists
    ✓ Should set alloc point (withUpdate)
    ✓ Should set alloc point
    ✓ Should emit SetPoolAllocPoint event
  #setMigrator()
```

```
    ✓ Should fail if caller is not owner
    ✓ Should set migrator address
    ✓ Should emit SetMigrator event
  #migrate()
    ✓ Should fail if there is no migrator
    ✓ Should fail if migrator failed to migrate to new lp token
    ✓ Should emit Migrate event
  #rewardDebt()
    ✓ Should fail if pid is invalid
    ✓ Should return reward debt
  #pendingMSpace()
    ✓ Should fail if pid is invalid
    ✓ Should return pending mspace
  #massUpdatePools()
    ✓ Should update all Pools
    ✓ Should emit MassUpdatePools event
  #updatePool()
    ✓ Should fail if pid is invalid
    ✓ Should update pool info and user balance
    ✓ Should emit UpdatePool event
  #updateUser()
    ✓ Should update user share
  #deposit()
    ✓ Should fail if deposit amount is invalid
    ✓ Should update user balance
    ✓ Should emit Deposit event
  #harvest()
    ✓ Should transfer MSpace token
    ✓ Should emit Harvest event
  #withdraw()
    ✓ Should fail if withdrawal amount exceed balance
    ✓ Should transfer LP token
    ✓ Should emit Withdraw event
  #emergencyWithdraw()
    ✓ Should transfer LP token
    ✓ Should emit EmergencyWithdraw event

MiningPoolMigrator
  #constructor()
    ✓ Should initalize contract params
  #migrate()
    ✓ Should fail if caller is not gravity
    ✓ Should fail if try to migrate too early (not before block)
    ✓ Should fail if try to migrate from old one
    ✓ Should return migrated pair
    ✓ Should update lp token balance

Extractor
  #init()
```

```
    ✓ Should initalize contract params
    ✓ Should fail if already initalized
  #bridgeFor()
    ✓ Should return wmeta address if token address is zero address
    ✓ Should return bridge address
  #setBridge()
    ✓ Should fail if caller is not owner
    ✓ Should fail if token address is invalid
    ✓ Should update bridge
    ✓ Should emit LogBridgeSet event
  #convert()
    ✓ Should fail if caller is not EOA
    ✓ Should fail if pair doesn't exist
    ✓ Should update lp token balance
    ✓ Should convert all token to mspace (MSP only)
    ✓ Should convert all token to mspace (META only)
    ✓ Should convert all token to mspace (USDT only)
    ✓ Should convert all token to mspace (MSP-META)
    ✓ Should convert all token to mspace (MSP-USDT)
    ✓ Should convert all token to mspace (META-USDT)
    ✓ Should convert all token to mspace (USDT-META)
    ✓ Should convert all token to mspace (MIC - USDT)
    ✓ Should convert all token to mspace (WBTC - DSD)
    ✓ Should convert all token to mspace (USDC - DSD)
    ✓ Should emit LogConvert event
  #convertMultiple()
    ✓ Should fail if caller is not EOA
    ✓ Should fail if pair doesn't exist
    ✓ Should convert multiple cases

MSpaceMintPlan
  #setStrategy()
    ✓ Should fail if caller is not owner
    ✓ Should update strategy
    ✓ Should emit UpdateConfig event
  #setStrategyPoint()
    ✓ Should fail if caller is not owner
    ✓ Should fail if point index is invalid
    ✓ Should update strategy point
    ✓ Should emit UpdateConfig event
  #setBlockStep()
    ✓ Should fail if caller is not owner
    ✓ Should update blockstep
    ✓ Should emit UpdateConfig event
  #strategyLength()
    ✓ Should return strategy length
  #mspacePerBlock()
    ✓ Should return reward per block (before start reward)
    ✓ Should return reward per block (before blockstep)
```

```
    ✓ Should return reward per block
  #calcReward()
    ✓ Should fail if input params are invalid
    ✓ Should return calculated reward (before start reward)
    ✓ Should return calculated reward (after all rewardStrategy)
    ✓ Should return calculated reward

MSpaceToken
  #init()
    ✓ Should initialize contract params
    ✓ Should fail if already initialized
  #mint()
    ✓ Should fail if caller is not owner
    ✓ Should mint token
  #setTeam()
    ✓ Should fail if caller is not team
    ✓ Should update team address
  #airdrop()
    ✓ Should fail if airdropYear is larger than 4
    ✓ Should fail if yearsPassed is zero
    ✓ Should mint to team

Bank
  #init()
    ✓ Should initialize contract params
    ✓ Should fail if already initialized
  #enter()
    ✓ Should locks MSP and mints xMSP (initial)
    ✓ Should locks MSP and mints xMSP
  #leave()
    ✓ Should unlocks MSP and burns xMSP

UniswapV2ERC20
  #UniswapV2ERC20_init()
    ✓ Should initialize contract params
  #_mint()
    ✓ Should update token balance
    ✓ Should emit Transfer event
  #_burn()
    ✓ Should update token balance
    ✓ Should emit Transfer event
  #approve()
    ✓ Should update token allowance
    ✓ Should emit Approval event
  #transfer()
    ✓ Should update token balance
    ✓ Should emit Transfer event
  #transferFrom()
    ✓ Should update token balance
```

```
    ✓ Should emit Transfer event
  #permit()
    ✓ Should fail if transaction expired
    ✓ Should fail if signature is invalid
    ✓ Should approve as signature

UniswapV2Factory
  #init()
    ✓ Should initialize contract params
    ✓ Should fail if already initialized
  #setPairImplement()
    ✓ Should fail if caller is not owner
    ✓ Should fail if pair address is zero address
    ✓ Should update pair implement
  #upgradePair()
    ✓ Should fail if pairProxy is not exist
  #lockPair()
    ✓ Should fail if caller is not owner
    ✓ Should fail if pair address is zero address
    ✓ Should lock pair
  #unlockPair()
    ✓ Should fail if caller is not owner
    ✓ Should fail if pair address is zero address
    ✓ Should unlock pair
  #allPairsLength()
    ✓ Should return pair list length
  #createPair()
    ✓ Should fail if tokenA and tokenB address is identical
    ✓ Should fail if token address is zero address
    ✓ Should fail if pair already exists
    ✓ Should create pair
    ✓ Should emit PairCreated event
  #setFeeTo()
    ✓ Should fail if caller is not owner
    ✓ Should update feeTo
  #setMigrator()
    ✓ Should fail if caller is not owner
    ✓ Should update migrator address

UniswapV2Migrator
  #constructor()
    ✓ Should initialize contract params

UniswapV2Pair
  #getReserves()
    ✓ Should return reserve data
  #initialize()
    ✓ Should initialize contract params
    ✓ Should fail if already initialized
```

```
  #version()
    ✓ Should return contract version
  #setLock()
    ✓ Should fail if caller is not owner
    ✓ Should update lock state
  #mint()
    ✓ Should fail if locked
    ✓ Should fail if reenter to contract
    ✓ Should fail minted liquidity is insufficient
    ✓ Should update LP token balance
    ✓ Should emit Mint event
  #burn()
    ✓ Should fail if locked
    ✓ Should fail if reenter to contract
    ✓ Should fail if totalSupply is zero
    ✓ Should fail if burned liquidity is too small
    ✓ Should update LP token balance and pair token balance
    ✓ Should emit Burn event
  #swap()
    ✓ Should fail if locked
    ✓ Should fail if reenter to contract
    ✓ Should fail if output amount is invaild
    ✓ Should fail if liquidity is insufficient
    ✓ Should fail if to address is token0 or token1 address
    ✓ Should fail if input amount is insufficient
    ✓ Should update token balance and reserves
    ✓ Should emit Swap event
  #skim()
    ✓ Should fail if locked
    ✓ Should fail if reenter to contract
    ✓ Should transfer all reserves to specified account
  #sync()
    ✓ Should fail if locked
    ✓ Should fail if reenter to contract
    ✓ Should update reserves and priceCummulative
    ✓ Should emit Sync event

UniswapV2Router02
  #init()
    ✓ Should initialize contract params
    ✓ Should fail if already initialized
  #addLiquidity()
    ✓ Should fail if deadline is expired
    ✓ Should fail if amountAOptimal is less than minAmount
    ✓ Should fail if amountBOptimal is less than minAmount
    ✓ Should update token balance
    ✓ Should update userInfo
  #addLiquidityMETA()
    ✓ Should fail if deadline is expired
```

```
  ✓ Should fail if WMETA transfer failed
  ✓ Should fail if amountAOptimal is less than minAmount
  ✓ Should fail if amountBOptimal is less than minAmount
  ✓ Should update token balance
  ✓ Should update META balance
  ✓ Should update userInfo
#removeLiquidity()
  ✓ Should fail if deadline is expired
  ✓ Should fail if amountA is less than minAmount
  ✓ Should fail if amountB is less than minAmount
  ✓ Should update token balance
  ✓ Should update userInfo
#removeLiquidityMETA()
  ✓ Should fail if deadline is expired
  ✓ Should fail if amountA is less than minAmount
  ✓ Should fail if amountB is less than minAmount
  ✓ Should update token balance
  ✓ Should update META balance
  ✓ Should fail if META transfer failed
  ✓ Should update userInfo
#removeLiquidityWithPermit()
  ✓ Should remove liquidity with permit
#removeLiquidityMETAWithPermit()
  ✓ Should remove liquidity with permit
#removeLiquidityMETASupportingFeeOnTransferTokens()
  ✓ Should fail if deadline is expired
  ✓ Should fail if amountA is less than minAmount
  ✓ Should fail if amountB is less than minAmount
  ✓ Should update token balance
  ✓ Should update META balance
  ✓ Should fail if META transfer failed
  ✓ Should update userInfo
#removeLiquidityMETAWithPermitSupportingFeeOnTransferTokens()
  ✓ Should remove liquidity with permit
#swapExactTokensForTokens()
  ✓ Should fail if deadline is expired
  ✓ Should fail if output amount is smaller than minimum amount
  ✓ Should swap tokens
#swapTokensForExactTokens()
  ✓ Should fail if deadline is expired
  ✓ Should fail if input amount exceeds maximum amount
  ✓ Should swap tokens
#swapExactMETAForTokens()
  ✓ Should fail if deadline is expired
  ✓ Should fail if first path is not WMETA
  ✓ Should fail if WMETA transfer failed
  ✓ Should fail if output amount is smaller than minimum amount
#swapTokensForExactMETA()
  ✓ Should fail if deadline is expired
```

```
  ✓ Should fail if last path is not WKLAY
  ✓ Should fail if WMETA transfer failed
  ✓ Should fail if input amount exceeds maximum amount
#swapExactTokensForMETA()
  ✓ Should fail if deadline is expired
  ✓ Should fail if last path is not WMETA
  ✓ Should fail if WMETA transfer failed
  ✓ Should fail if output amount is smaller than minimum amount
#swapMETAForExactTokens()
  ✓ Should fail if deadline is expired
  ✓ Should fail if first path is not WMETA
  ✓ Should fail if WMETA transfer failed
  ✓ Should fail if input amount exceeds maximum amount
#swapExactTokensForTokensSupportingFeeOnTransferTokens()
  ✓ Should fail if deadline is expired
  ✓ Should fail if output amount is smaller than minimum amount
  ✓ Should swap tokens
#swapExactMETAForTokensSupportingFeeOnTransferTokens()
  ✓ Should fail if deadline is expired
  ✓ Should fail if first path is not WMETA
  ✓ Should fail if WMETA transfer failed
  ✓ Should fail if output amount is smaller than minimum amount
#swapExactTokensForMETASupportingFeeOnTransferTokens()
  ✓ Should fail if deadline is expired
  ✓ Should fail if last path is not WMETA
  ✓ Should fail if WMETA transfer failed
  ✓ Should fail if output amount is smaller than minimum amount
#quote()
  ✓ Should return quote
#getAmountOut()
  ✓ Should return output amount
#getAmountIn()
  ✓ Should return input amount
#getAmountsOut()
  ✓ Should return output amounts
#getAmountsIn()
  ✓ Should return input amounts
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
/contracts/dao					
Timelock.sol	100	100	100	100	
/contracts/extra					
WMETA.sol	100	100	100	100	

/contracts/helpers					
BancorConverterRegistry.sol	100	100	100	100	
BoringHelper.sol	100	100	100	100	
ENSRegistry.sol	100	100	100	100	
ENSRegistryWithFallback.sol	100	100	100	100	
/contracts/helpers/interfaces					
ENS.sol	100	100	100	100	
IERC20.sol	100	100	100	100	
IFactory.sol	100	100	100	100	
IMiningHill.sol	100	100	100	100	
IOracle.sol	100	100	100	100	
IPair.sol	100	100	100	100	
IStrategy.sol	100	100	100	100	
Multicall2.sol	100	100	100	100	
/contracts/helpers/libraries					
BoringERC20.sol	100	100	100	100	
BoringMath.sol	100	100	100	100	
BoringPair.sol	100	100	100	100	
/contracts/helpers/utility					
IOwned.sol	100	100	100	100	
Owned.sol	100	100	100	100	
Utils.sol	100	100	100	100	
/contracts/proxy					
UpgradeableProxy.sol	100	100	100	100	

/contracts/stake					
EternalStorage.sol	100	100	100	100	
MiningHill.sol	100	100	100	100	
Extractor.sol	100	100	100	100	
MSpaceMintPlan.sol	100	100	100	100	
MSpaceToken.sol	100	100	100	100	
Bank.sol	100	100	100	100	
/contracts/stake/interfaces					
IMSpace.sol	100	100	100	100	
/contracts/stake/libraries					
SafeERC20.sol	100	100	100	100	
SafeMath.sol	100	100	100	100	
/contracts/uniswapv2/interfaces					
IERC20Uniswap.sol	100	100	100	100	
IUniswapV2Callee.sol	100	100	100	100	
IUniswapV2ERC20.sol	100	100	100	100	
IUniswapV2Factory.sol	100	100	100	100	
IUniswapV2Pair.sol	100	100	100	100	
IUniswapV2Router01.sol	100	100	100	100	
IUniswapV2Router02.sol	100	100	100	100	
IWMETA.sol	100	100	100	100	
/contracts/uniswapv2/libraries					
Math.sol	100	100	100	100	
SafeMathUniswap.sol	100	100	100	100	

TransferHelper.sol	100	100	100	100	
UniswapV2Library.sol	100	100	100	100	
UQ112x112.sol	100	100	100	100	
/contracts/uniswapv2					
UniswapV2ERC20.sol	100	100	100	100	
UniswapV2Factory.sol	100	100	100	100	
UniswapV2Migrator.sol	100	100	100	100	
UniswapV2Pair.sol	100	100	100	100	
UniswapV2Router02.sol	100	100	100	100	

[Table 1] Test Case Coverage

End of Document