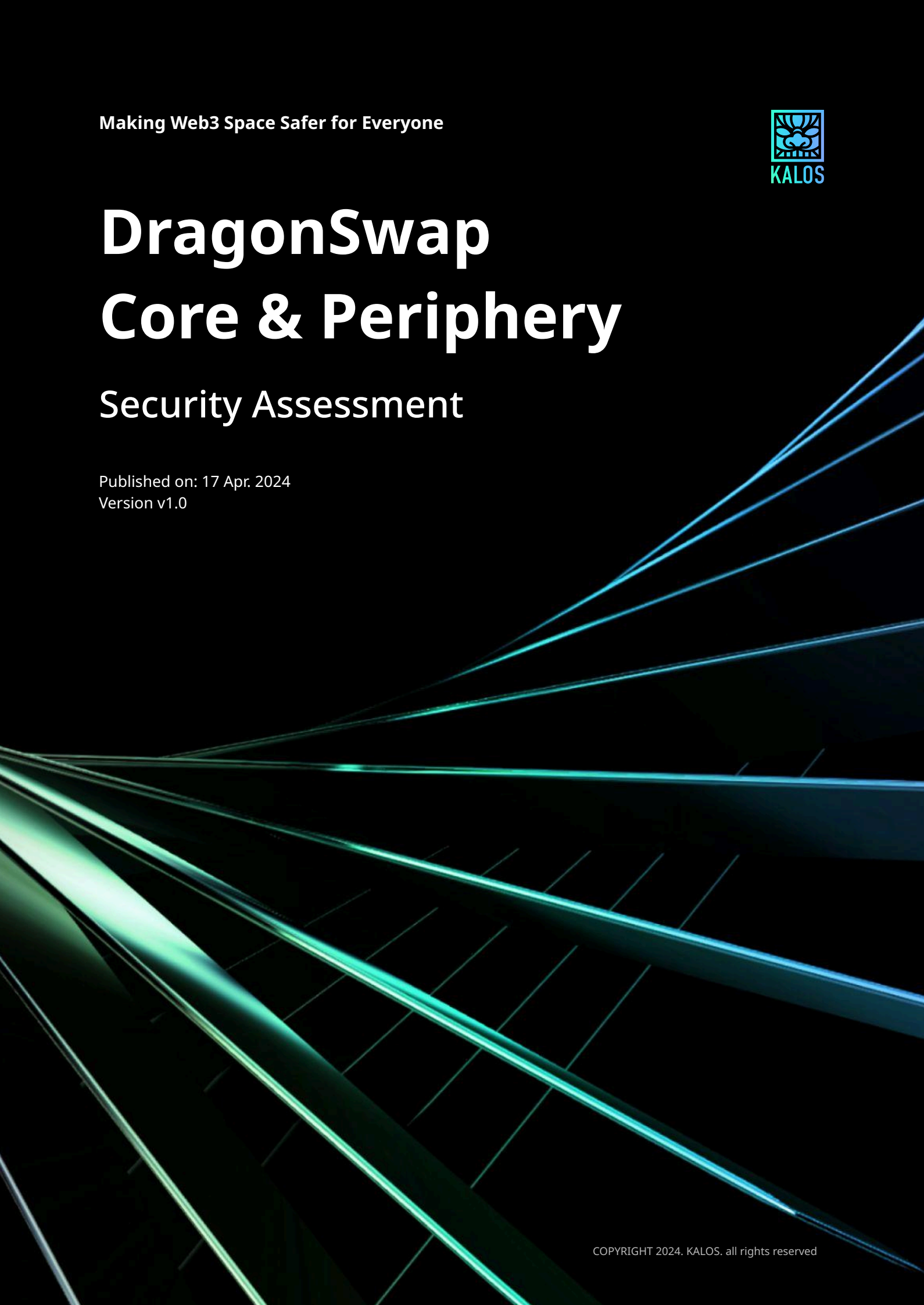
Making Web3 Space Safer for Everyone

KALOS

DragonSwap Core & Periphery

Security Assessment

Published on: 17 Apr. 2024
Version v1.0



Security Report Published by KALOS

v1.0 17 Apr. 2024

Found issues

Severity of Issues	Findings	Resolved	Acknowledged	Comment
Critical	-		-	-
High	-	-	-	-
Medium	-	-	-	-
Low	1	1	-	-
Tips	4	-	4	-

TABLE OF CONTENTS

ABOUT US

Making Web3 Space Safer for Everyone

KALOS is a flagship service of HAECHI LABS, the leader of the global blockchain industry. We bring together the best Web2 and Web3 experts. Security Researchers with expertise in cryptography, leaders of the global best hacker team, and blockchain/smart contract experts are responsible for securing your Web3 service.

Having secured $60B crypto assets on over 400 main-nets, Defi protocols, NFT services, P2E, and Bridges, KALOS 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@kalos.xyz
Website: https://kalos.xyz

Executive Summary

Purpose of this report

This report was prepared to audit the security of Dragon Swap's AMM DEX and staking contracts, which are forks of the PancakeSwap. KALOS conducted the audit focusing on whether the system created by Dragon Swap is soundly implemented and designed as specified in the published materials, in addition to the safety and security of the AMM DEX and staking contracts.
In detail, we have focused on the following

- Project availability issues like Denial of Service
- Strict access control on storage variables to prevent unauthorized access
- Function access control measures
- Reward miscalculation
- Risk from fee and tick spacing modification
- Arithmetic error

Codebase Submitted for the Audit

The codes used in this Audit can be found on GitHub

- https://github.com/dragon-swap-klaytn/dragon-smart-contracts

The last commit of the code used for this Audit is

- d92a40a2261244aa682eb540fe1e94f4d5278b30

The last commit of the code patched for this Audit is

- bad8f6d7ba6cfc9800dc9de6561f483906bd5c13

Audit Timeline

Date	Event
2024/04/03	Audit Initiation
2024/04/17	Delivery of v1.0 report.

Findings

KALOS found 1 Low severity and 4 Tips issues that would improve the code’s usability or efficiency upon modification.

Severity	Issue	Status
Tips	1. Fee tier in the implementation does not match the fee in the document	(Acknowledged - v1.0)
Tips	2. Denial of Service in Permit of SwapRouter	(Acknowledged - v1.0)
Low	3. Unnecessary function leads to less fee distribution	(Resolved - v1.0)
Tips	4. Outdated MultiSigWallet	(Acknowledged - v1.0)
Tips	5. Centralization Risk of arbitrary call in factory owner	(Acknowledged - v1.0)

OVERVIEW

Protocol overview

• **AMM Pool**

The Automated Market Maker Pool contracts are a fork of the PancakePair and PancakeV3Pool contracts. The v2-core contracts implement a CPMM pool, and the v3-core contracts implement a concentrated liquidity pool. The DragonSwap team adjusted the protocol fee rate of both v2 and v3 pools. Furthermore, the swap fee tier of the v3-core contract varies as the tick spacing of the pool set is adjusted. The pools are deployed using factory contracts. The owner of the factory contract is capable of collecting the protocol fee accumulated on the pools and setting the LM Pool of the pools.

• **Swap Router**

The router contract serves as a liaison between the user and the pool contracts. It enables a multi-hop swap that user can swap a token with another token even if the pair pool doesn't exist. The router contract also checks if the slippage the user can bear is actually satisfied during the swap process. Additionally, this router does not support stable swap.

• **Non-Fungible Token Position Manager**

On the concentrated liquidity pools, users can mint and burn the position NFTs through the NonfungiblePositionManager contract. As the user's position data is inscribed in the NFT, the user can send or receive the pool's position by transferring the NFT.
This contract enables the user to mint and burn the pool's position and increase and decrease the liquidity of the position, and also, collect the swap fees accumulated on the position.

• **Master Chef**

Users can deposit the position NFT to the Master Chef V3 contract or LP tokens to the V2 contract. The owner of the Master Chef contract sets a reward emission rate and each pool's reward weight. As the time passed, the staked LP tokens or NFTs accrue the reward. The accumulated rewards distributed to the users pro rata to their share of liquidity and the weight of the pool the position made. The V2 contract calculates the liquidity shares

based on the total liquidity and the V3 contract calculates the shares on the basis that the liquidity actually used (based on the current tick).

• **LM Pool**

The PancakeV3LmPool contract used by the MasterChefV3 contract. This contract tracks the pool’s tick movement such that it calculates the growth reward amounts and the user position’s reward amounts.

Scope

masterchef-v2/contracts

```
├── MasterChefV2.sol
└── interfaces
    └── IERC20Metadata.sol
```

masterchef-v3/contracts

```
├── Enumerable.sol
├── MasterChefV3.sol
├── interfaces
│   ├── IFarmBooster.sol
│   ├── ILMPool.sol
│   ├── ILMPoolDeployer.sol
│   ├── IMasterChefV2.sol
│   ├── IMasterChefV3.sol
│   ├── INonfungiblePositionManager.sol
│   ├── INonfungiblePositionManagerStruct.sol
│   ├── IPancakeV3Pool.sol
│   ├── IReceiver.sol
│   └── IWETH.sol
├── libraries
│   └── SafeCast.sol
└── utils
    └── Multicall.sol
```

v2-core/contracts

```
├── PancakeERC20.sol
├── PancakeFactory.sol
├── PancakePair.sol
├── PancakeRouter.sol
├── PancakeRouter01.sol
├── PancakeZapV1.sol
├── interfaces
│   ├── IERC20.sol
│   ├── IPancakeCallee.sol
│   ├── IPancakeERC20.sol
│   ├── IPancakeFactory.sol
│   ├── IPancakeMigrator.sol
│   ├── IPancakePair.sol
```

```
│   ├── IPancakeRouter01.sol
│   ├── IPancakeRouter02.sol
│   └── IWETH.sol
├── libraries
    ├── Babylonian.sol
    ├── Math.sol
    ├── PancakeLibrary.sol
    ├── SafeMath.sol
    ├── UQ112x112.sol
    └── WBNB.sol
```

v3-core/contracts

```
├── PancakeV3Factory.sol
├── PancakeV3Pool.sol
├── PancakeV3PoolDeployer.sol
├── interfaces
│   ├── IERC20Minimal.sol
│   ├── IPancakeV3Factory.sol
│   ├── IPancakeV3Pool.sol
│   ├── IPancakeV3PoolDeployer.sol
│   ├── callback
│   │   ├── IPancakeV3FlashCallback.sol
│   │   ├── IPancakeV3MintCallback.sol
│   │   └── IPancakeV3SwapCallback.sol
│   └── pool
│       ├── IPancakeV3PoolActions.sol
│       ├── IPancakeV3PoolDerivedState.sol
│       ├── IPancakeV3PoolEvents.sol
│       ├── IPancakeV3PoolImmutables.sol
│       ├── IPancakeV3PoolOwnerActions.sol
│       └── IPancakeV3PoolState.sol
└── libraries
    ├── BitMath.sol
    ├── FixedPoint128.sol
    ├── FixedPoint96.sol
    ├── FullMath.sol
    ├── LiquidityMath.sol
    ├── LowGasSafeMath.sol
    ├── Oracle.sol
    ├── Position.sol
    ├── SafeCast.sol
```

```
    ├── SqrtPriceMath.sol
    ├── SwapMath.sol
    ├── Tick.sol
    ├── TickBitmap.sol
    ├── TickMath.sol
    ├── TransferHelper.sol
    └── UnsafeMath.sol
```

v3-lm-pool/contracts

```
├── PancakeV3LmPool.sol
├── PancakeV3LmPoolDeployer.sol
├── interfaces
│   ├── ILMPool.sol
│   ├── IMasterChefV3.sol
│   └── IPancakeV3LmPool.sol
└── libraries
    └── LmTick.sol
```

v3-periphery/contracts

```
├── NFTDescriptorEx.sol
├── NonfungiblePositionManager.sol
├── NonfungibleTokenPositionDescriptor.sol
├── NonfungibleTokenPositionDescriptorOffChain.sol
├── NonfungibleTokenPositionDescriptorOffChainV2.sol
├── PancakeInterfaceMulticallV2.sol
├── SwapRouter.sol
├── V3Migrator.sol
├── base
│   ├── BlockTimestamp.sol
│   ├── ERC721Permit.sol
│   ├── LiquidityManagement.sol
│   ├── Multicall.sol
│   ├── PeripheryImmutableState.sol
│   ├── PeripheryPayments.sol
│   ├── PeripheryPaymentsWithFee.sol
│   ├── PeripheryValidation.sol
│   ├── PoolInitializer.sol
│   └── SelfPermit.sol
├── interfaces
```

```
│   ├── IERC20Metadata.sol
│   ├── IERC721Permit.sol
│   ├── IMulticall.sol
│   ├── INonfungiblePositionManager.sol
│   ├── INonfungibleTokenPositionDescriptor.sol
│   ├── IPeripheryImmutableState.sol
│   ├── IPeripheryPayments.sol
│   ├── IPeripheryPaymentsWithFee.sol
│   ├── IPoolInitializer.sol
│   ├── IQuoter.sol
│   ├── IQuoterV2.sol
│   ├── ISelfPermit.sol
│   ├── ISwapRouter.sol
│   ├── ITickLens.sol
│   ├── IV3Migrator.sol
│   └── external
│       ├── IERC1271.sol
│       ├── IERC20PermitAllowed.sol
│       └── IWETH9.sol
├── lens
│   ├── PancakeInterfaceMulticall.sol
│   ├── Quoter.sol
│   ├── QuoterV2.sol
│   └── TickLens.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
```

v3-router/contracts

```
├── FactoryOwner.sol
├── MultiSigWallet.sol
├── SmartRouter.sol
├── V2SwapRouter.sol
├── V3SwapRouter.sol
├── base
│   ├── ApproveAndCall.sol
│   ├── ImmutableState.sol
│   ├── MulticallExtended.sol
│   ├── OracleSlippage.sol
│   ├── PeripheryPaymentsExtended.sol
│   ├── PeripheryPaymentsWithFeeExtended.sol
│   └── PeripheryValidationExtended.sol
├── interfaces
│   ├── IApproveAndCall.sol
│   ├── IImmutableState.sol
│   ├── IMixedRouteQuoterV1.sol
│   ├── IMulticallExtended.sol
│   ├── IOracleSlippage.sol
│   ├── IPeripheryPaymentsExtended.sol
│   ├── IPeripheryPaymentsWithFeeExtended.sol
│   ├── IQuoter.sol
│   ├── IQuoterV2.sol
│   ├── ISmartRouter.sol
│   ├── ITokenValidator.sol
│   ├── IV2SwapRouter.sol
│   ├── IV3SwapRouter.sol
│   └── IWETH.sol
├── lens
│   ├── MixedRouteQuoterV1.sol
│   ├── Quoter.sol
│   ├── QuoterV2.sol
│   └── TokenValidator.sol
└── libraries
    ├── Constants.sol
    ├── PoolTicksCounter.sol
    └── SmartRouterHelper.sol
```

Access Controls

- MasterChef V2
  - Owner: Updates cake amount per block, pool info, whitelist, boost contract address.
  - Boost Contract: Updates boost multiplier.

- MasterChef V3
  - Owner: Updates emergency status, period duration receiver/lm pool deployer/operator/boost contract address, pool info.
  - Operator: Accumulates reward.
  - Receiver: Changes latest period info.
  - Boost Contract: Updates boost multiplier.

- V2 Core
  - feeToSetter: Updates feeTo (Protocol Fee) and feeToSetter address.
  - PancakeZapV1 Owner: Updates max zap reverse ratio value and can recover wrong tokens.

- V3 Core
  - Factory
    - Owner: Enables fee amount info, updates whitelist addresses, lm pool info, fee protocol info, and collects protocol fee.
    - LmPool Deployer: Updates lm pool info.
  - Pool
    - Factory and Factory Owner : Modifies protocol fee and collects accumulated protocol fee.

- V3 Periphery:
  - NFTDescriptorEx Owner : Updates switchToHttpLink and NFTDomain.

- V3 Lm Pool
    - Pool: Liquidity and reward growth global values are changed.
    - Master Chef: Updates position and deploys lm pool.

- V3 Router
  - FactoryOwner

    - Owner: Updates operators, fee receivers, and arbitrarily calls external contracts.
    - Operator: Removes liquidity from v2 pools and collects/distributes protocol fees.
- MultiSigWallet
    - Operator: Add transactions to tx list.

Each privileged account has permissions that can change the crucial part of the system. Currently, all of these owners of the contracts are EOA. It is highly recommended to maintain the private key as securely as possible and strictly monitor the system state changes.

FINDINGS

1. Fee tier in the implementation does not match the fee in the document

ID: DRAGON-01
Severity: Tips
Type: Documentation
Difficulty: Low
File: v3-core/contracts/PancakeV3Factory.sol

Issue

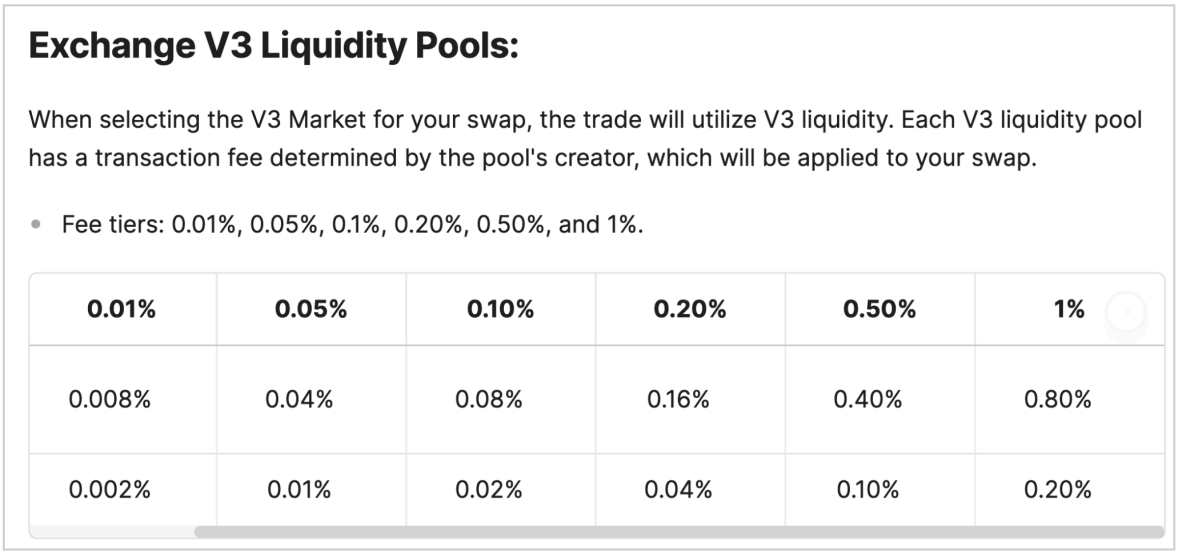
The implementation does not support the 0.1% fee tier although the fee specified in the document.

```
constructor(address _poolDeployer) {
    poolDeployer = _poolDeployer;
    owner = msg.sender;
    emit OwnerChanged(address(0), msg.sender);

    feeAmountTickSpacing[100] = 1;
    feeAmountTickSpacingExtraInfo[100] = TickSpacingExtraInfo({whitelistRequested: false, enabled:
true});
    emit FeeAmountEnabled(100, 1);
    emit FeeAmountExtraInfoUpdated(100, false, true);
    feeAmountTickSpacing[500] = 10;
    feeAmountTickSpacingExtraInfo[500] = TickSpacingExtraInfo({whitelistRequested: false, enabled:
true});
    emit FeeAmountEnabled(500, 10);
    emit FeeAmountExtraInfoUpdated(500, false, true);
    feeAmountTickSpacing[2000] = 40;
    feeAmountTickSpacingExtraInfo[2000] = TickSpacingExtraInfo({whitelistRequested: false, enabled:
true});
    emit FeeAmountEnabled(2000, 40);
    emit FeeAmountExtraInfoUpdated(2000, false, true);
    feeAmountTickSpacing[5000] = 100;
    feeAmountTickSpacingExtraInfo[5000] = TickSpacingExtraInfo({whitelistRequested: false, enabled:
true});
    emit FeeAmountEnabled(5000, 100);
    emit FeeAmountExtraInfoUpdated(5000, false, true);
    feeAmountTickSpacing[10000] = 200;
    feeAmountTickSpacingExtraInfo[10000] = TickSpacingExtraInfo({whitelistRequested: false, enabled:
true});
    emit FeeAmountEnabled(10000, 200);
    emit FeeAmountExtraInfoUpdated(10000, false, true);
}
```

[https://github.com/dragon-swap-klaytn/dragon-smart-contracts/blob/d92a40a2261244aa682eb540fe1e94f4d5278b30/projects/v3-core/contracts/PancakeV3Factory.sol#L36-L61]

The current implementation supports only 0.01%, 0.05%, 0.2%, 0.5%, and 1% fee tiers without 0.1%.



Exchange V3 Liquidity Pools:

When selecting the V3 Market for your swap, the trade will utilize V3 liquidity. Each V3 liquidity pool has a transaction fee determined by the pool's creator, which will be applied to your swap.

- Fee tiers: 0.01%, 0.05%, 0.1%, 0.20%, 0.50%, and 1%.

0.01%	0.05%	0.10%	0.20%	0.50%	1%
0.008%	0.04%	0.08%	0.16%	0.40%	0.80%
0.002%	0.01%	0.02%	0.04%	0.10%	0.20%

[The Fee tier stated in the document]

Recommendation

Add the 0.1% fee tier to the factory.

2. Denial of Service in Permit of SwapRouter

ID: DRAGON-02
Type: Documentation
File: v3-periphery/base/SelfPermit.sol

Severity: Tips
Difficulty: Low

Issue

The permit function in the SelfPermit contract can be front-run to cause a user's transaction to fail (DoS). The signature (v, r, s) parameters for the permit call can be searched in the mempool and can be used by another contract. Since the signature can be used only once, calling the increaseLiquidity, decreaseLiquidity, and mint functions that use the permit function can fail.

```
abstract contract SelfPermit is ISelfPermit {
    /// @inheritdoc ISelfPermit
    function selfPermit(
        address token,
        uint256 value,
        uint256 deadline,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) public payable override {
        IERC20Permit(token).permit(msg.sender, address(this), value, deadline, v, r, s);
    }
```

[https://github.com/dragon-swap-klaytn/dragon-smart-contracts/blob/d92a40a2261244aa682eb540fe1e94f4d5278b30/projects/v3-periphery/contracts/base/SelfPermit.sol]

Recommendation

Document the risk of the SelfPermit contract to prevent misuse of the functions.

3. Unnecessary function leads to less fee distribution

ID: DRAGON-03
Type: Documentation
File: v3-router/contracts/FactoryOwner.sol

Severity: Low
Difficulty: High

Issue

Unlike removeFeeReceiver(), remove() does not decrease totalWeight. If this function is called, the distribute() will distribute less fee, and the remaining fee will not be distributed.

```
function removeFeeReceiver(address _receiver) external onlyOwner {
    uint256 len = feeReceiverLength();
    uint256 index = type(uint256).max;
    for (uint256 i = 0; i < len; i++) {
        if (feeReceiver[i].receiver == _receiver) {
            index = i;
            break;
        }
    }

    if (index != type(uint256).max) {
        totalWeight = totalWeight - feeReceiver[index].weight;
        feeReceiver[index] = feeReceiver[len - 1];
        feeReceiver.pop();

        emit RemoveFeeReceiver(_receiver, totalWeight);
    }
}

function remove() external onlyOwner {
    feeReceiver.pop();
}
```

[https://github.com/dragon-swap-klaytn/dragon-smart-contracts/blob/d92a40a2261244aa682eb540fe1e94f4d5278b30/projects/v3-router/contracts/FactoryOwner.sol#L154-L175]

Recommendation

Remove remove() function

Fix Comment

[d74fceaa] commit removed the function.

4. Outdated MultiSigWallet

ID: DRAGON-04
Severity: Tips
Type: Documentation
Difficulty: Low
File: v3-router/contracts/MultiSigWallet.sol

Issue

The current MultiSigWallet implementation is outdated.

Recommendation

Use the latest version of multi-sig wallet.

Fix Comment

This contract will not be used.

5. Centralization Risk of arbitrary call in factory owner

ID: DRAGON-05	Severity: Tips
Type: Documentation	Difficulty: Low
File: v3-router/contracts/FactoryOwner.sol	

Issue

This contract has an arbitrary call function. It allows an owner to withdraw the deposited LP tokens and the collected fee without using the intended functions.

```
function execute(address _to, uint256 _value, bytes memory _data) external onlyOwner {
    (bool result,) = _to.call{value: _value}(_data);
    if (!result) {
        revert();
    }
}
```

[https://github.com/dragon-swap-klaytn/dragon-smart-contracts/blob/main/projects/v3-router/contracts/FactoryOwner.sol#L123-L128]

Recommendation

Remove execute() function and add event emission for monitoring

Fix Comment

The team replied they will not fix.

[bad8f6d7] commit added event emission.

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 on the main network. In order to write secure codes, correction of discovered problems and sufficient testing thereof are required.

Appendix. A

Severity Level

CRITICAL	Must be addressed as a vulnerability that has the potential to seize or freeze substantial sums of money.
HIGH	Has to be fixed since it has the potential to deny users compensation or momentarily freeze assets.
MEDIUM	Vulnerabilities that could halt services, such as DoS and Out-of-Gas, need to be addressed.
LOW	Issues that do not comply with standards or return incorrect values
TIPS	Tips that makes the code more usable or efficient when modified

Difficulty Level

	Low	Medium	High
Privilege	anyone	Miner/Block Proposer	Admin/Owner
Capital needed	Small or none	Gas fee or volatile as price change	More than exploited amount
Probability	100%	Depend on environment	Hard as mining difficulty

Vulnerability Category

Arithmetic	• Integer under/overflow vulnerability • floating point and rounding accuracy
Access & Privilege Control	• Manager functions for emergency handle • Crucial function and data access • Count of calling important task, contract state change, intentional task delay
Denial of Service	• Unexpected revert handling • Gas limit excess due to unpredictable implementation
Miner Manipulation	• Dependency on the block number or timestamp. • Frontrunning
Reentrancy	•Proper use of Check-Effect-Interact pattern. •Prevention of state change after external call • Error handling and logging.
Low-level Call	• Code injection using delegatecall • Inappropriate use of assembly code
Off-standard	• Deviate from standards that can be an obstacle of interoperability.
Input Validation	• Lack of validation on inputs.
Logic Error/Bug	• Unintended execution leads to error.
Documentation	•Coherency between the documented spec and implementation
Visibility	• Variable and function visibility setting
Incorrect Interface	• Contract interface is properly implemented on code.

End of Document