yAudit Wen markets Review

Review Resources:

- None beyond the code repository

Auditors:

- Drastic Watermelon
- Panda

Table of Contents

Review Summary

Wen markets

Wen markets provide a set of smart contracts to simplify permissionless token deployment and liquidity bootstrapping for users.

The contracts of the Wen markets Repo were reviewed over five days. The code review was performed by two auditors between the 1st of July and the 5th of July 2024. The repository was not under active development during the review, and the review was limited to commit 6340b7d94e45e4c38f640da5d289a3fd187ef149 for the Wen markets repo.

Scope

The scope of the review consisted of the following contracts at the specific commit:

```
src/
├── WenFoundry.sol
├── WenHeadmaster.sol
├── WenLedger.sol
├── WenToken.sol
├── periphery/
├──── WenRouter.sol
```

After the findings were presented to the Wen markets team, fixes were made and included in several PRs.

This review is a code review to identify potential vulnerabilities in the code. The reviewers did not investigate security practices or operational security and assumed that privileged accounts could be trusted. The reviewers did not evaluate the security of the code relative to a standard or specification. The review may not have identified all potential attack vectors or areas of vulnerability.

yAudit and the auditors make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAudit and the auditors do not represent nor imply to third parties that the code has been audited nor that the code is free from defects. By deploying or using the code, Wen markets and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	Proper mechanisms are in place to ensure that only authorized users can perform sensitive operations. However, additional checks and balances, such as the one recommended to avoid ownership renouncement while paused, could further enhance security.
Mathematics	Good	The mathematical operations and formulas used in the contracts are sound. The suggestion to use unchecked math for gas savings is noted but not critical for L1s or L2s, as stated by the developers.

Category	Mark	Description
Complexity	Good	The code complexity is manageable, and the logic is clear. Some refactoring, such as inline modifiers, could further reduce complexity and improve readability.
Libraries	Good	Using external libraries is appropriate and contributes to code robustness.
Decentralization	Good	The protocol's design supports decentralized operations.
Code stability	Good	No changes were made to the scope of the contracts during the review.
Documentation	Low	The protocol lacks external documentation regarding its core business processes. The code presents NATSPEC documentation only for some external methods.
Monitoring	Average	The code was found to lack event emissions for crucial state changes, compromising the effectiveness of off-chain monitoring solutions. The WenLedger contract presents a central aggregation point for some of the protocol's core insights, although a possible DOS issue caused by OOG exceptions, even within an eth_call, was identified.
Testing and verification	Low	Given the low coverage provided by the test suite, there is a need for more rigorous testing and verification, with a particular focus on achieving high code coverage. The test suite may also be improved by implementing basic fuzz or invariant tests for core invariant properties.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact
  - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements.
- Gas savings

    - Findings that can improve the gas efficiency of the contracts.
- Informational
    - Findings including recommendations and best practices.

Critical Findings

None

High Findings

1. High - Instantaneous update of the K value will trigger arbitrage opportunities

With the K_ value changing atomically, an arbitrage bot could sandwich the transaction for profit.

Technical Details

```
91 |    function setInitVirtualEthReserve(uint256 initVirtualEthReserve) external
onlyOwner {
92 |        initVirtualEthReserve_ = initVirtualEthReserve;
93 |        K_ = initVirtualEthReserve_ * INIT_VIRTUAL_TOKEN_RESERVE;
94 |        graduationThreshold_ = K_ / (INIT_VIRTUAL_TOKEN_RESERVE -
INIT_REAL_TOKEN_RESERVE) - initVirtualEthReserve_;
95 |    }
```

Impact

High. Arbitrage will extract value from the protocol.

Recommendation

Add K to the pool structure and never change the value once a pool is created.

Developer Response

Added K to the `Pool` struct in 7a78c96.

2. High - Token graduation can be perpetually DOS'd

`Headmaster.execute()` is responsible for executing a token's graduation, which involves allowing token users to make approvals for the token, creating a UniswapV2 pool for the token at hand and WMATIC, and seeding such pool with an initial amount of liquidity. The whole graduation process can be forced to fail by creating the UniswapV2 pool before the token's graduation.

Technical Details

Because anyone can deploy a UniswapV2 pool for any two addresses UniswapV2Factory.sol#L23 and this process does not require the deployer to add an initial amount of liquidity to the pool, an attacker can permanently DOS a `WenToken`'s graduation process by simply deploying the pool himself.

Find a working PoC here

Impact

High. Perpetual DOS of protocol's core functionality.

Recommendation

Avoid attempting to deploy the pool if it already exists:

```
function execute(WenToken token, uint256 amountToken, uint256 amountEth)
        external
        payable
        onlyWenFoundry
        returns (uint256 poolId, uint256 _amountToken, uint256 _amountETH)
{
        if (amountToken == 0) revert InvalidAmountToken();
        if (amountEth == 0 || msg.value != amountEth) revert InvalidAmountEth();

        SafeTransferLib.safeTransferFrom(token, msg.sender, address(this), amountToken);
        SafeTransferLib.safeApprove(token, address(uniswapV2Router02), amountToken);

-       address pair = uniswapV2Factory.createPair(address(token),
uniswapV2Router02.WETH());
+       address pair = uniswapV2Factory.getPair(address(token),
uniswapV2Router02.WETH());
+       if (pair == address(0)) pair = uniswapV2Factory.createPair(address(token),
uniswapV2Router02.WETH());
        poolId = uint256(uint160(pair));
        (_amountToken, _amountETH,) = uniswapV2Router02.addLiquidityETH{ value: amountEth
}(address(token), amountToken, 0, 0, liquidityOwner, block.timestamp);

        alumni.push(token);

        emit Executed(token, poolId, _amountToken, _amountETH, liquidityOwner);
}
```

Developer Response

Fixed in `8703b14` as part of https://github.com/wenmarkets-org/wen-contracts-private/pull/4

3. High - WenTokens are tradeable on DEXs before graduation

`WenToken`s are intended to pass through a "graduation" process, by which upon reaching a certain market cap, the token "graduates" and obtains the ability to be traded on UniswapV2-like DEXs. Furthermore, this process deploys an initial UniswapV2-like pool on such DEX, seeding it with initial liquidity. Before it goes through this upgrade, a `WenToken` is solely intended to be traded via the `WenFoundry` contract, which implements a common bonding curve for all tokens, whose proceeds are used as the initial liquidity during the "graduation" process.

Technical Details

The protocol intends to prohibit users from trading `WenToken`s on other avenues by prohibiting them from setting a non-zero allowance to any contract other than `WenFoundry`. With this, no contract should be able to pull tokens from a user and thus should not be able to use them in a swap. This restriction doesn't take into account the way that UniswapV2Pool's low-level methods function: `mint()`, `swap()` and `burn()` do not pull funds from a user, rather they expect the user's funds to be already within the contract. As a consequence, with these methods users and third-party contracts can get past the limitation imposed by the protocol and effectively trade `WenToken`s on DEXs before their "graduation".

Find a working PoC here

Impact

High. Core protocol invariant can be easily breached.

Recommendation

The protocol should create a token's pair as the token's creation occurs, or at least precompute the pair's address and blacklist transfers to such address. Upon graduation, the blacklist may be lifted to allow the protocol to supply the initial liquidity and users to swap freely in the pool.

Developer Response

Fixed in `56707f0` as part of https://github.com/wenmarkets-org/wen-contracts-private/pull/5

Medium Findings

None

Low Findings

1. Low - `approve` will always revert if the `IERC20` interface mismatch

Some tokens, such as USDT, have a different implementation for the approve function. When the address is cast to a compliant IERC20 interface, and the approve function is used, it will always revert due to the interface mismatch.

Technical Details

```
File: src/periphery/WenRouter.sol

192 | tokenIn.approve(address(router), _amountIn);

257 | tokenIn.approve(address(router), amountIn);
```

src/periphery/WenRouter.sol#L192, src/periphery/WenRouter.sol#L257

Impact

Low.

Recommendation

Use safeERC20 approve functions.

Developer Response

Replaced these with Solmate's `SafeTransferLib` in `b4266fc7`.

2. Low - The owner can renounce ownership while the system is paused

The contract owner is not prevented from renouncing the ownership while the contract is paused, which would cause any user assets stored in the protocol to be locked indefinitely.

Technical Details

The WenFoundry swapping function is pause protected, the owner of the contract could pause and renounce ownership of the contract, it will block the funds present in the contract.

Impact

Low.

Recommendation

Make sure ownership can't be changed when the contract is paused.

Developer Response

Added check in 30c908c6

3. Low - creationFee should be capped

Although the `creationFee` can be set to any value, it's advised to cap the max fee at a reasonable value.

Technical Details

```
97 |    function setCreationFee(uint256 fee) external onlyOwner {
98 |        creationFee_ = fee;
99 |    }
```

src/WenFoundry.sol#L97-L99

Impact

Low.

Recommendation

Cap the creation fee to a reasonable value.

Developer Response

Acknowledged. This is left as-is, because there is no concrete plan if this fee should be set at all or not. If turned on, the creation fee is also charged upfront, so the users can easily notice the amount of Ether they are paying before they sign the transaction.

4. Low - Several fields will eventually cause out-of-gas exceptions

Several dynamic arrays in different contracts are only ever appended to. Because these fields are only readable via getters that return the whole array, the gas cost for invoking such methods will grow monotonically and eventually lead to OOG issues.

Technical Details

Following is a list of dynamic arrays that can only be appended to and the getters which may incur OOG exceptions:

- WenHeadmaster.sol#L24, WenHeadmaster.sol#L64-L66
- WenToken.sol#L31, WenToken.sol#L107-L118, WenToken.sol#L123-L125
- WenLedger.sol#L38, WenLedger.sol#L102-L104
- WenLedger.sol#L39, WenLedger.sol#L114-L116
- WenLedger.sol#L42, WenLedger.sol#L126-L128
- WenLedger.sol#L43, WenLedger.sol#L142-L144
- WenLedger.sol#L44, WenLedger.sol#L150-L152

Most importantly, notice that this issue also applies in the case in which the above-mentioned methods are intended to be exclusively used by off-chain components, given that, generally, nodes of an EVM chain can define a custom gas limit for eth_call JSON RPC calls.

Impact

Low. The calling cost of certain getter methods will increase monotonically over time, eventually making it impossible to invoke them both on-chain and off-chain.

Recommendation

Implement paginated reads for the read arrays: the getter methods should specify the starting and ending indexes of elements to be fetched. This way, off-chain components can split reads between various eth_call RPC requests, and on-chain components can limit the number of elements retrieved to a number that will not cause the transaction to incur a large cost or revert by running out of gas.

Developer Response

Added pagination and optimized array related reads in 63ff58ac, d153408, and 0b28df1, which are all included in https://github.com/wenmarkets-org/wen-contracts-private/pull/5. The protocol never calls these methods in order to function, and all data is also indexed off-chain. The array view functions are only convenience methods consumed on the frontend as a hybrid approach to reduce load on the indexer.

5. Low - Referrals are modifiable

Referrals can be gamed to cheat referrers out of their fee.

Technical Details

Because a user's referral can be indefinitely set and cleared by using `WenRouter.setReferrer()` and `WenRouter.resetReferrer()`, users can cheat their referrer out of its expected fee by resetting their referrer address.

Impact

Low. Referrers can be cheated out of their rewards.

Recommendation

Make referrals immutable: users should be able to set them at most once.

Developer Response

Acknowledged. This is by design, the users are allowed to change their referrer addresses.

6. Low - Incorrect referral logs

The events emitted by `WenRouter._resolveReferrer()` are incorrect in 2 separate ways.

Technical Details

1. Both `AddReferer` and `RemoveReferer` events define the first parameter to be account whose referral is set, and the second parameter to be the referrer. The event emissions shown at WenRouter.sol#L131 and WenRouter.sol#L133 are provided the parameters in the wrong order.
2. The `RemoveReferer` log at WenRouter.sol#L133 logs the incorrect referrer account: it should log that the storage value `referrer_[msg.sender]` is being removed, not the calldata param `referrer`.

Impact

Low. Incorrect event emissions will compromise off-chain monitoring efforts.

Recommendation

1. Correct the order of the parameters provided to both logs
2. Modify the parameter passed to `RemoveReferer` to reflect the actual action occurring. Consider refactoring the event to a `ChangedReferer` where both the old and new referer are logged.

Developer Response

Fixed and refactored to use a single event `ChangeReferrer` in `aec369da`

7. Low - Anyone can pull stray funds left in `WenRouter`

`WenRouter` implements a wrapper for a common UniswapV2Router02, adding a referral feature and an additional fee for users swapping through the protocol's custom router.

Technical Details

Because `WenRouter` doesn't store the reference UniswapV2-like router in storage but receives it as calldata for every method it implements, and because no validation is enforced on such calldata parameter, any user can specify a custom contract as the `router` parameter for `swapExactTokensForTokens()` and `swapExactTokensForEth()` methods.

Such a contract will be granted approval for `amountIn` in the selected input token and can also return `0` as the output amount of tokens when invoked by the `WenRouter` contract, avoiding having to pay any fees. Thus, the malicious router can pull stray funds left in the `WenRouter` contract, either within the same transaction or in a separate one.

Impact

Low. Funds sent mistakenly to `WenRouter` can be pulled by anyone.

Recommendation

Enforce adequate verification on the `router` parameter for all swap methods in `WenRouter`. If it is known that only specific routers will be used throughout the protocol's lifespan, store this information as an immutable variable in the contract; otherwise, implement a router whitelist.

Developer Response

Fixed by implementing a router whitelist and a pair of fund rescue methods in 1571f3b6 and 51e6819b, included in https://github.com/wenmarkets-org/wen-contracts-private/pull/4. The router is not designed to hold funds, so these are implemented purely to protect a user's mistake.

Gas Saving Findings

1. Gas - Caching global variables is more expensive than using the actual variable

It's better not to cache global variables, as their direct usage is cheaper.

Technical Details

```
File: src/periphery/WenRouter.sol

218 | uint256 amountIn = msg.value;
```

src/periphery/WenRouter.sol#L218

Impact

Gas savings.

Recommendation

Do not create a variable for msg.value.

Developer Response

Acknowledged. The current and future versions of the protocol will only be deployed on gas-cheap L1s or L2s, so we will skip the gas optimization for simplicity and readability for now, since assigning an amountIn is more readable here.

2. Gas - Use unchecked math

Unchecked math can be used for the following operations.

Technical Details

```
File: src/WenFoundry.sol
274 | uint256 newVirtualTokenReserve = K_ / newVirtualEthReserve;

321 | uint256 newVirtualEthReserve = K_ / newVirtualTokenReserve;

360 | uint256 newVirtualEthReserve = K_ / newVirtualTokenReserve;

378 | uint256 newVirtualTokenReserve = K_ / newVirtualEthReserve;
```

src/WenFoundry.sol#L274, src/WenFoundry.sol#L321, src/WenFoundry.sol#L360, src/WenFoundry.sol#L378

Impact

Gas savings

Recommendation

Use unsafe math.

```
unchecked {
    uint256 newVirtualTokenReserve = K_ / newVirtualEthReserve;
}
```

Developer Response

Acknowledged. The current and future versions of the protocol will only be deployed on gas-cheap L1s or L2s, so we will skip the gas optimization for simplicity and readability for now.

3. Gas - Inline `modifiers` that are only used once to save gas

Consider removing the following modifiers and put the logic directly in the function where they are used, as they are used only once.

Technical Details

```
File: src/WenHeadmaster.sol


32 | modifier onlyWenFoundry() {
33 |         if (msg.sender != address(wenFoundry)) revert Forbidden();
34 |         _;
35 |     }
```

src/WenHeadmaster.sol#L32

Impact

Gas savings

Recommendation

Inline the modifier into the function.

Developer Response

Acknowledged. The current and future versions of the protocol will only be deployed on gas-cheap L1s or L2s, so we will skip the gas optimization for simplicity and readability for now, since naming this check is more readable.

4. Gas - Check the non-SLOAD condition first

Some conditions may be reordered to save a SLOAD (2100 gas). Avoid reading state variables when the first part of the condition fails in a successful or (`||`) statement.

Technical Details

```
File: src/periphery/WenRouter.sol

179 | bool isBuy = settlementTokens_[tokenIn] || tokenIn == WETH;

280 | bool isBuy = settlementTokens_[tokenIn] || tokenIn == WETH;
```

src/periphery/WenRouter.sol#L179, src/periphery/WenRouter.sol#L280

Impact

Gas savings.

Recommendation

Change the order in the condition.

Developer Response

Refactored in `7caaeb27`.

5. Gas - Structs can be packed into fewer storage slots

Each slot saved can avoid an extra Gsset (**20000 gas**) for the first setting of the struct.

Technical Details

```
File: src/WenLedger.sol


// @audit: 1 slot could be saved, by using the following order
struct Trade {
  uint256 amountIn; // (256 bits)
   uint256 amountOut; // (256 bits)
   contract WenToken token; // (160 bits)
   bool isBuy; // (8 bits)
   address maker; // (160 bits)
   uint128 timestamp; // (128 bits)
   uint128 blockNumber; // (128 bits)
}
```

src/WenLedger.sol#L21

Impact

Gas savings.

Recommendation

Change the structure order a.

Developer Response

Implemented in `ee29a24`.

6. Gas - `graduate` could transfer the tokens instead of approving `headmaster`

The current approach allows the headmaster to pull funds from the foundry to the headmaster via a transferFrom at a later point in time. A direct transfer can be done instead to save gas.

Technical Details

```
File: src/WenFoundry.sol
223 |         token.approve(address(headmaster_), type(uint256).max);
```

src/WenFoundry.sol#L223 src/WenHeadmaster.sol#L48

Impact

Gas savings.

Recommendation

```
- token.approve(address(headmaster_), type(uint256).max);
+ token.transfer(address(headmaster_), _amountToken);
```

```
-  SafeTransferLib.safeTransferFrom(token, msg.sender, address(this), amountToken);
```

Developer Response

Acknowledged and won't fix. The current and future versions of the protocol will only be deployed on gas-cheap L1s or L2s, so we will skip the gas optimization for simplicity and readability for now.

Informational Findings

1. Informational - Events emission is missing

State variables changes must emit an event.

Technical Details

File: src/WenFoundry.sol

```
77 | function setFeeTo(address feeTo) external onlyOwner { // @audit no events emitted.
78 |         feeTo_ = feeTo;
79 |     }

81 | function setFeeRate(uint256 feeRate) external onlyOwner {
82 |         if (feeRate > MAX_FEE) revert FeeTooHigh();
83 |         feeRate_ = feeRate;
84 |     }

86 | function setGraduationFeeRate(uint256 feeRate) external onlyOwner {
87 |         if (feeRate > MAX_FEE) revert FeeTooHigh();
88 |         graduationFeeRate_ = feeRate;
89 |     }

91 | function setInitVirtualEthReserve(uint256 initVirtualEthReserve) external onlyOwner
{
92 |         initVirtualEthReserve_ = initVirtualEthReserve;
93 |         K_ = initVirtualEthReserve_ * INIT_VIRTUAL_TOKEN_RESERVE;
94 |         graduationThreshold_ = K_ / (INIT_VIRTUAL_TOKEN_RESERVE -
INIT_REAL_TOKEN_RESERVE) - initVirtualEthReserve_;
95 |     }

97 | function setCreationFee(uint256 fee) external onlyOwner {
98 |         creationFee_ = fee;
99 |     }

101 | function setHeadmaster(WenHeadmaster headmaster) external onlyOwner {
102 |          headmaster_ = headmaster;
103 |     }

105 | function setOwner(address owner) external onlyOwner {
106 |          owner_ = owner;
```

```
107 |     }

109 | function setPaused(bool paused) external onlyOwner {
110 |         paused_ = paused;
111 |     }
```

src/WenFoundry.sol#L77, src/WenFoundry.sol#L81, src/WenFoundry.sol#L86, src/WenFoundry.sol#L91, src/WenFoundry.sol#L97, src/WenFoundry.sol#L101, src/WenFoundry.sol#L105, src/WenFoundry.sol#L109

File: src/periphery/WenRouter.sol

```
47 | function setOwner(address owner) external onlyOwner {
48 |         owner_ = owner;
49 |     }

67 | function setCanPause(address account, bool canPause) external onlyOwner {
68 |         emergencyPausers_[account] = canPause;
69 |     }

71 | function setFeeTo(address feeTo) external onlyOwner {
72 |         feeTo_ = feeTo;
73 |     }

75 | function setFeeRate(uint256 feeRate) external onlyOwner {
76 |         if (feeRate > MAX_FEE) revert FeeTooHigh();
77 |         feeRate_ = feeRate;
78 |     }

80 | function setReferralRate(uint256 referralRate) external onlyOwner {
81 |         if (referralRate > MAX_FEE) revert FeeTooHigh();
82 |         referralRate_ = referralRate;
83 |     }

85 | function setSettlementToken(ERC20 token, bool isSettlementToken) external onlyOwner
{
86 |         settlementTokens_[token] = isSettlementToken;
87 |     }
```

src/periphery/WenRouter.sol#L47, src/periphery/WenRouter.sol#L67, src/periphery/WenRouter.sol#L71, src/periphery/WenRouter.sol#L75, src/periphery/WenRouter.sol#L80, src/periphery/WenRouter.sol#L85

Impact

Informational.

Recommendation

Add events for state variable changes.

Developer Response

Added in ece65f42.

2. Informational - `if`-statement can be converted to a ternary

The code can be made more compact while also increasing readability by converting the following `if`-statements to ternaries (e.g., `foo += (x > y) ? a : b`)

Technical Details

```
File: src/periphery/WenRouter.sol


137 | if (referrer_[msg.sender] != address(0)) {
138 |                 _referrer = referrer_[msg.sender];
139 |             } else {
140 |                 _referrer = feeTo_;
141 |             }
```

src/periphery/WenRouter.sol#L137

Impact

Informational.

Recommendation

Use a ternary.

Developer Response

Acknowledged. This is a personal preference where the if-statements are chosen in this case because the conditions are a bit long.

3. Informational - Unused import

The identifier is imported but never used within the file.

Technical Details

```
File: src/periphery/WenRouter.sol

9 | import { WenToken } from "../WenToken.sol";

10 | import { WenHeadmaster } from "../WenHeadmaster.sol";

11 | import { WenLedger } from "../WenLedger.sol";
```

src/periphery/WenRouter.sol#L9, src/periphery/WenRouter.sol#L10, src/periphery/WenRouter.sol#L11

Impact

Informational.

Recommendation

Remove the unused imports.

Developer Response

Removed in 74f3e7ca

4. Informational - Unused library

`WenRouter` imports and uses solmate's `FixedPointMathLib` but fails to use any of its methods.

Technical Details

No methods from the library cited above are used within the contract.

WenRouter.sol#L24

Impact

Informational.

Recommendation

Remove the unused library.

Developer Response

Fixed. The unused library is removed in `0e7fabf3`.

5. Informational - Missing whitelist for `WenToken` instances

The protocol fails to verify that the traded tokens are instances of the `WenToken` contract.

Technical Details

Both `WenFoundry` and `WenRouter` never verify that a token entering or exiting a swap is a token deployed by the protocol.

Impact

Informational.

Recommendation

Implement a whitelist only to allow trades swapping from or to a `WenToken`. Such a whitelist should be populated upon token creation and checked before a user's swap is executed.

Developer Response

Whitelist check is implemented in `WenFoundry.sol` via `a95fa4f6` and included in https://github.com/wenmarkets-org/wen-contracts-private/pull/5. However, on `WenRounter.sol`, we don't perform checks like this by design because we intend to allow any types of swaps via the router.

Final remarks

The audit revealed several critical areas requiring attention to ensure the protocol's security and efficiency. While the protocol demonstrates good practices in access control, mathematical accuracy, complexity management, and the use of libraries, there are notable deficiencies in documentation, monitoring, and testing. Addressing high-severity issues such as arbitrage opportunities and potential DoS attacks, improving event emissions, and implementing a more rigorous testing framework are essential steps forward. Enhancing these aspects will significantly strengthen the protocol's stability, security, and overall performance, thereby fostering greater trust and reliability among users.