Metastreet Contracts v2

Security Review

Cantina Managed review by:
Desmond Ho, Lead Security Researcher
Defsec, Security Researcher

April 30, 2024

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

MetaStreet v2, also known as "the automatic tranche maker" (ATM) is a permissionless lending protocol for on-chain collateral. Pools are organized by automatically tranching capital based on different risk and rate profiles from depositors, which turns into fixed-duration loans for borrowers. The main goals of The ATM is to improve on three shortcomings of existing lending protocols: oracleless (remove the dependency on a centralized price oracle for loan to value limits), dynamic interest rate model (replace a fixed, governance-driven interest rate model with a dynamic, deposit driven one) and permissionless (allow users to instantiate a lending pool for any collection permissionlessly).

From Mar 27th to Apr 9th the Cantina team conducted a review of metastreet-contracts-v2 on commit hash c01152ac. The team identified a total of **13** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 1
- Medium Risk: 0
- Low Risk: 2
- Gas Optimizations: 0
- Informational: 10

3 Findings

3.1 High risk

3.1.1 Claimable `BANANA` may exceed wrapper's balance

Severity: High risk

Context: KongzBundleCollateralWrapper.sol#L329-L332

Description: There is an edge case where the claimable amount can exceed the balance claimed by the contract, due to rounding issues.

The main reason is that `_lastUpdatedTimestamps` is updated in `YieldHub` for all Kongz held in this wrapper contract (per user accounting), but the accounting here is updating only for the bundle, thus having a discrepancy.

Due to solidity rounding down, the sum of the division of constituents may be lesser than the division of the entire component: eg. (`7/3 > 1/3 * 7`). Hence, the time delta between each `_claim()` will be smaller or equal to that for a bundle. Larger bundles are more susceptible to this problem than smaller ones as they take up a larger portion of the total Kongz held by the wrapper. The impact would be the failure to claim BANANAs and unwrap Kongz.

Proof of concept:

Assume 2 bundles:

- Small bundle (1 Kongz) claiming every 1236 seconds (103 blocks).
- Large bundle (6 Kongz).

After just 3 claims, large Kongz bundle will revert due to the rounding issue.

- Small bundle banana amount each claim: `1 * 1236 * 1e19 / 86400 = 143055555555555555` .
- Claim for whole contract each time: `7 * 1236 * 1e19 / 86400 = 1001388888888888888` .
- Total claimed for contract after 3 claims: `1001388888888888888 * 3 = 3004166666666666664` .
- Total claimed by small bundle after 3 claims: `143055555555555555 * 3 = 429166666666666665`
- Remaining claimable balance: `3004166666666666664 - 429166666666666665 = 2574999999999999999` .
- Large bundle claimable balance: `6 * 1236 * 3 * 1e19 / 86400 = 2575000000000000000`, 1 wei more than the claimable balance.

Note that the test below requires the `allowBlocksWithSameTimestamp` to be set to `true` in `hardhat.config.ts`:

```
networks: {
    hardhat: {
      allowUnlimitedContractSize: true,
      allowBlocksWithSameTimestamp: true,
      chainId: 1,
    },
    //...
}
```

Add the following test to `KongzBundleCollateralWrapper.spec.ts`:

```
describe.only("#revert claimable", async function () {
    let contextSmallBundle: string;
    let contextLargeBundle: string;
    let tokenIdSmallBundle: ethers.BigNumber;
    let tokenIdLargeBundle: ethers.BigNumber;
    const KONGZ_ID_7 = ethers.BigNumber.from("522");

    it("will cause claim to fail due to rounding", async function () {
      let currentTime = await helpers.time.latest() + 1;
      await helpers.time.setNextBlockTimestamp(currentTime);
      const mintTxLargeBundle = await kongzBundleCollateralWrapper
        .connect(accountBorrower)
```

```
        .mint([KONGZ_ID_2, KONGZ_ID_3, KONGZ_ID_4, KONGZ_ID_5, KONGZ_ID_6, KONGZ_ID_7] /* To mint large bundle
↪  */);
      tokenIdLargeBundle = (await extractEvent(mintTxLargeBundle, kongzBundleCollateralWrapper,
↪  "BundleMinted")).args.tokenId;
      await helpers.time.setNextBlockTimestamp(currentTime);
      const mintTxSmallBundle = await kongzBundleCollateralWrapper.connect(accountBorrower).mint([KONGZ_ID_1]);
      tokenIdSmallBundle = (await extractEvent(mintTxSmallBundle, kongzBundleCollateralWrapper,
↪  "BundleMinted")).args.tokenId;

      /* Create context */
      contextLargeBundle = ethers.utils.solidityPack(
        ["address", "uint256[]"],
        [accountBorrower.address, [KONGZ_ID_2, KONGZ_ID_3, KONGZ_ID_4, KONGZ_ID_5, KONGZ_ID_6, KONGZ_ID_7]]
      );

      contextSmallBundle = ethers.utils.solidityPack(
        ["address", "uint256[]"],
        [accountBorrower.address, [KONGZ_ID_1]]
      );

      let interval = 1236;
      currentTime = await helpers.time.latest();

      // claim small bundle 3 times
      for (let i = 0; i < 3; ++i) {
        currentTime += interval;
        await helpers.time.setNextBlockTimestamp(currentTime);
        await kongzBundleCollateralWrapper.claim(tokenIdSmallBundle, contextSmallBundle);
      }

      // now claim large bundle, but will revert
      await kongzBundleCollateralWrapper.claim(tokenIdLargeBundle, contextLargeBundle);
    });
  })
```

Recommendation: Cap the transfer amount to the `BANANA` amount held by the contract:

```
if (amount > 0) {
  // or import and use OZ.Math.min()
  amount = _banana.balanceOf(address(this)) >= amount ? amount : _banana.balanceOf(address(this));
  _banana.transfer(minter, amount);
}
```

Metastreet: Resolved in commit 29240cf3.

Cantina Managed: Fix is verified. Also, confirmed through the proof of concept provided.

3.2 Low risk

3.2.1 Incorrect storage slot calculation

Severity: Low risk

Context: Pool.sol#L172-L172

Description: The `erc7201` prefix is incorrectly included as part of the `NAMESPACE_ID` for the storage slot calculation.

Description: Unfortunately, the production version is v2.6 which has this erroneous calculation. Hence, the calculated slot may already be in use by existing loans. It is recommended to document the error, but leave it as is.

Metastreet: Resolved in commit 29321adb.

Cantina Managed: Issue resolved by adding comment.

3.2.2 Lack of validation for quote price in `_verifyQuote` function

Severity: Low risk

Context: SimpleSignedPriceOracle.sol#L153-L153

Description: During the code review of the `_verifyQuote` function, it was identified that there is no validation check to ensure that the `quote.price` is greater than zero. Allowing quotes with zero prices could result in unintended economic consequences or exploitations within the system. This lack of validation undermines the robustness of the quote verification process and could potentially be exploited by malicious actors to disrupt normal operations.

Recommendation: It is recommended to introduce a validation check within the `_verifyQuote` function to ensure that quote.price is greater than zero.

Metastreet: Resolved in commit c62db088.

Cantina Managed: Fix is verified.

3.3 Informational

3.3.1 Potential failure in unwrapping ERC1155 tokens with blacklisting or pause feature

Severity: Informational

Context: ERC1155CollateralWrapper.sol#L263-L263

Description: The provided `unwrap` function in the smart contract is responsible for burning the ERC1155CollateralWrapper token and transferring the underlying ERC1155 tokens back to the token owner. However, this function assumes that the `safeBatchTransferFrom` call to the ERC1155 token contract will always succeed.

If the underlying ERC1155 token contract has implemented blacklisting or pause features, the `safeBatchTransferFrom` call may revert or fail under certain conditions, such as:

- **Blacklisting:** If the recipient address (in this case, msg.sender) is blacklisted by the ERC1155 token contract, the transfer will fail.
- **Pause Feature:** If the ERC1155 token contract has a pause functionality, and the contract is currently paused, the transfer will fail.

In such cases, the unwrap function will not be able to complete successfully, potentially leading to stuck tokens or lost funds.

```
/**
  * Emits a {BatchUnwrapped} event
  *
  * @inheritdoc ICollateralWrapper
  */
function unwrap(uint256 tokenId, bytes calldata context) external nonReentrant {
    if (tokenId != uint256(_hash(context))) revert InvalidContext();
    if (msg.sender != ownerOf(tokenId)) revert InvalidCaller();

    /* Decode context */
    (address token, , , uint256[] memory tokenIds, uint256[] memory quantities) = abi.decode(
        context,
        (address, uint256, uint256, uint256[], uint256[])
    );

    /* Burn ERC1155CollateralWrapper token */
    _burn(tokenId);

    /* Batch transfer tokens back to token owner */
    IERC1155(token).safeBatchTransferFrom(address(this), msg.sender, tokenIds, quantities, "");

    emit BatchUnwrapped(tokenId, msg.sender);
}
```

Recommendation: Introduce checks before the integration with blacklisteable/pausable tokens.

Metastreet: Acknowledged.

Cantina Managed: Acknowledged.

3.3.2 Explicitly state return arguments in `_price()`

Severity: Informational

Context: WeightedInterestRateModel.sol#L54-L54

Description: Consider explicitly stating the return arguments to prevent accidental mix-ups for future code changes.

Recommendation:

```
- ) internal pure override returns (uint256, uint256) {
+ ) internal pure override returns (uint256 repayment, uint256 adminFee) {
```

Metastreet: Acknowledged. The returns are documented in the `InterestRateModel` abstract contract for the time being. We've opted to leave the returns unchanged for now in `WeightedInterestRateModel`

Cantina Managed: Acknowledged.

3.3.3 Restrictive zero oracle context requirement

Severity: Informational

Context: ExternalPriceOracle.sol#L84-L85

Description: There could be a use-case where an oracle doesn't require additional context to determine the price of the requested collateral, such as the `ChainlinkPriceOracle`. In this scenario, it would be necessary to pass dummy data, unnecessarily wasting gas.

Recommendation: Consider allowing zero context.

```
- /* Return 0 if oracle context does not exist */
- if (oracleContext.length == 0) return 0;

  // Modify comment in Pool.sol
- /* Get oracle price if price oracle and oracle context exist, else 0 */
+ /* Get oracle price if price oracle exists, else 0 */
```

MetaStreet: Resolved in commit 2437c291.

Cantina Managed: Fixed.

3.3.4 Redundant `IPriceOracle` import

Severity: Informational

Context: Pool.sol#L25-L25

Description: Redundant import.

Recommendation:

```
- import "./interfaces/IPriceOracle.sol";
```

Metastreet Resolved in commit 07b7238b.

Cantina Managed: Fix is verified.

3.3.5 Incorrect comment

Severity: Informational

Context: KongzBundleCollateralWrapper.sol#L457-L457

Description: Incorrect comment.

Recommendation:

```
- /* Calculate claimable BANANA and send to minter */
+ /* Calculate claimable BANANA */
```

Metastreet: Resolved in commit 5d428573.

Cantina Managed: Fix is verified.

3.3.6 Missing `node` natspec

Severity: Informational

Context: LiquidityLogic.sol#L397-L398

Description: Natspec is missing for the `node` input parameter.

Recommendation:

```
+ * @param node Liquidity node
```

Metastreet: Resolved in commit 4f9b1a48.

Cantina Managed: Fix is verified.

3.3.7 Enhancement for `transferCalldata` in `ERC1155CollateralWrapper`

Severity: Informational

Context: ERC1155CollateralWrapper.sol#L173-L173

Description: The `ERC1155CollateralWrapper` contract's `transferCalldata` function is currently designed to facilitate the transfer of a single token type. However, this function currently does not support scenarios involving multiple entry point tokens.

Sandbox Asset: 0x7fbf5c9af42a6d146dcc18762f515692cd5f853b

Recommendation: To address this limitation and enhance the contract's functionality, it is recommended to extend the `transferCalldata` function or introduce a new function capable of handling multiple entry point tokens.

Metastreet: Acknowledged.

Cantina Managed: Acknowledged.

3.3.8 Enhancement of `setSigner` function for batch Processing

Severity: Informational

Context: SimpleSignedPriceOracle.sol#L236-L236

Description: The current implementation of the `setSigner` function within the Admin API allows for setting a price oracle signer for a single collateral token at a time. This approach, while functional, may not be the most efficient, especially in scenarios requiring the update of signers for multiple collateral tokens. Processing each collateral token individually can be time-consuming and gas-inefficient, particularly for platforms dealing with a large number of tokens.

Recommendation: To improve efficiency and reduce operational costs, it is recommended to modify the setSigner function to accept arrays of collateral tokens and their corresponding signers. This batch processing approach will allow multiple signers to be updated in a single transaction, significantly reducing the gas costs associated with these operations and simplifying administrative tasks.

Metastreet: Acknowledged.

Cantina Managed: Acknowledged. We may implement this in a future version, if we find multiple signer updates to be common.

3.3.9 Replay attacks due to missing nonce in `_verifyQuote` function

Severity: Medium risk

Context: SimpleSignedPriceOracle.sol#L146-L146

Description: The `_verifyQuote` function in the contract lacks a nonce mechanism to prevent replay attacks. This function verifies quotes and signers for transactions but does not include a nonce, which is crucial for ensuring that each quote is unique and cannot be reused maliciously.

In the testing scenario described, quotes for collateral tokens are created and verified without any form of nonce or transaction count that would ensure their uniqueness. This omission allows for the potential replay of previously signed quotes, leading to unauthorized or unintended transactions.

Proof of Concept:

```
describe("#price", async function () {
  beforeEach("set signer for WPUNKs", async function () {
    await simpleSignedPriceOracle.setSigner(WPUNKS_ADDRESS, accounts[0].address);
  });

  it("successfully return price", async function () {
    const message_1 = await createSignedQuote(accounts[0], WPUNKS_ADDRESS, WPUNK_ID_1,
↪  ethers.utils.parseEther("2"));
    let oracleContext = ethers.utils.defaultAbiCoder.encode(
      ["((address,uint256,address,uint256,uint64,uint64),bytes)[]"],
      [[message_1]]
    );

    /* Fast forward 30 seconds */
    await helpers.time.increase(30);

    /* Validate for 1 collateral token ID */
    expect(
      await simpleSignedPriceOracle.price(WPUNKS_ADDRESS, WETH_ADDRESS, [WPUNK_ID_1], [1], oracleContext)
    ).to.be.equal(ethers.utils.parseEther("2"));

    const message_2 = await createSignedQuote(accounts[0], WPUNKS_ADDRESS, WPUNK_ID_2,
↪  ethers.utils.parseEther("4"));
    oracleContext = ethers.utils.defaultAbiCoder.encode(
      ["((address,uint256,address,uint256,uint64,uint64),bytes)[]"],
      [[message_1, message_2]]
    );

    /* Fast forward 30 seconds */
    await helpers.time.increase(30);

    /* Validate for 2 collateral token IDs */
    expect(
      await simpleSignedPriceOracle.price(
        WPUNKS_ADDRESS,
        WETH_ADDRESS,
        [WPUNK_ID_1, WPUNK_ID_2],
        [1, 1],
        oracleContext
      )
    ).to.be.equal(ethers.utils.parseEther("3"));

    /* Validate for 2 collateral token IDs with quantity 2 and 3 respectively */
    expect(
      await simpleSignedPriceOracle.price(
        WPUNKS_ADDRESS,
        WETH_ADDRESS,
        [WPUNK_ID_1, WPUNK_ID_2],
        [2, 3],
        oracleContext
      )
    ).to.be.equal(ethers.utils.parseEther("3.2"));


    await simpleSignedPriceOracle.price(
```

```
    WPUNKS_ADDRESS,
    WETH_ADDRESS,
    [WPUNK_ID_1, WPUNK_ID_2],
    [2, 3],
    oracleContext
  );
});
```

Recommendation: Implement a nonce mechanism within the quote signing and verification process. Each quote should include a unique nonce that is checked against a stored value in the contract.

Metastreet: Acknowledged. We expect signed quotes to be valid for multiple users, and would like to allow caching of quotes by price oracle providers. For our use cases, we assume loan origination won't affect the market price of the underlying collateral for relatively short time frames that the quotes are valid.

Cantina Managed: Acknowledged due to false positive.

3.3.10 Lack of two-step ownership transfer

Severity: Informational

Context: SimpleSignedPriceOracle.sol#L13-L13

Description: To change the owner address, the current contract owner can call the `Ownable.transferOwnership()` function and set a new address and this new address assumes the role immediately. If the new address is inactive or not willing to act in the role, there is no way to restore access to that role. Therefore, the owner role can be lost.

Recommendation: It is recommended to use the following Ownable2Step library instead of Ownable library.

Metastreet: Resolved in commit baba72f4.

Cantina Managed: Fix is verified.