MakerDAO: Spark ALM

Security Review

Cantina Managed review by:

Christoph Michel, Lead Security Researcher

M4rio.eth, Security Researcher

February 27, 2025

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

The Maker Protocol, also known as the Multi-Collateral Dai (MCD) system, allows users to generate Dai (a decentralized, unbiased, collateral-backed cryptocurrency soft-pegged to the US Dollar) by leveraging collateral assets approved by the Maker Governance, which is the community organized and operated process of managing the various aspects of the Maker Protocol.

From Feb 12th to Feb 17th the Cantina team conducted a review of spark-alm-controller-private on tags v1.2.0-beta.0 and v1.3.0-beta.0. The directories in scope for this review were:

- `src`
- `deploy`

The team identified a total of **7** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 1
- Low Risk: 1
- Gas Optimizations: 0
- Informational: 5

The Cantina team reviewed MakerDAO's `spark-alm-controller-private` holistically on tag v1.3.0 and concluded that all issues were addressed and no new vulnerabilities were identified.

3 Findings

3.1 Medium Risk

3.1.1 Malicious relayer can permanently lock BUIDL transfers & redemptions by creating many issuances

Severity: Medium Risk

Context: MainnetController.sol#L485-L488

Description: Minting BUIDL works by transferring USDC to an address and the BUIDL admin asynchronously issues BUIDL tokens to the minter by calling `buidl.issueTokens(almProxy, amount)`. Note that all issuances to an investor are tracked in the Compliance Service's storage:

```
// BUIDL.issueTokens -> issueTokensCustom -> issueTokensWithMultipleLocks -> TokenLibrary.issueTokensCustom
//  -> IDSComplianceService(_services[COMPLIANCE_SERVICE]).validateIssuance(_to, _value, _issuanceTime)
//  -> recordIssuance(_to, _value, issuanceTime)
//  -> createIssuanceInformation(getRegistryService().getInvestor(_to), _value, _issuanceTime)

function createIssuanceInformation(
    string memory _investor,
    uint256 _value,
    uint256 _issuanceTime
) internal returns (bool) {
    uint256 issuancesCount = issuancesCounters[_investor];

    issuancesValues[_investor][issuancesCount] = _value;
    issuancesTimestamps[_investor][issuancesCount] = _issuanceTime;
    issuancesCounters[_investor] = issuancesCount + 1;

    return true;
}
```

When redeeming BUIDL back to USDC, `buidlRedeem.redeem(usdcAmount)` is called which performs a BUIDL transfer from the redeemer to the contract. This triggers many checks through the `transferFrom`'s `canTransfer()` modifier. Some of the checks iterate over all issuances:

```
function getComplianceTransferableTokens(
    address _who,
    uint256 _time,
    uint64 _lockTime
) public view returns (uint256) {
    require(_time != 0, "Time must be greater than zero");

    string memory investor = getRegistryService().getInvestor(_who);
    uint256 balanceOfInvestor = getLockManager().getTransferableTokens(_who, _time);
    uint256 investorIssuancesCount = issuancesCounters[investor];

    uint256 totalLockedTokens = 0;
    for (uint256 i = 0; i < investorIssuancesCount; i++) {
        uint256 issuanceTimestamp = issuancesTimestamps[investor][i];

        if (_lockTime > _time || issuanceTimestamp > SafeMath.sub(_time, _lockTime)) {
            totalLockedTokens = totalLockedTokens + issuancesValues[investor][i];
        }
    }

    // ...

    return transferable;
}
```

A malicious relayer can trigger around 10,000 issuances and the iteration cost for a redemption would exceed the block gas limit of 30e6 gas (each iteration performs a cold `sload`). Note that there's no way to decrease the `issuancesCounters` again. The BUIDL tokens will be permanently locked unless the third-party upgrades their contracts.

Recommendation: Consider reaching an agreement with BUIDL such that all USDC transfers are batched in a single issuance over a time period, for example, 1 issuance per day for all transfers received to the dedicated address.

Proof of Concept: Add the following test to `spark-alm-controller/test/mainnet-fork/Buidl.t.sol`:

```
contract CantinaTest is MainnetControllerDepositRedeemBUIDLE2ESuccessTests {
    uint256 internal speedup = 10;

    function setUp() public virtual override {
        super.setUp();

        vm.label(address(0x0A65a40a4B2F64D3445A628aBcFC8128625483A4), "LOCK_MANAGER");
        vm.label(address(0x1dc378568cefD4596C5F9f9A14256D8250b56369), "COMPLIANCE_CONFIGURATION_SERVICE");
        vm.label(address(0x07A1EBFb9a9A421249DDC71Bddb8860cc077E3a9), "COMPLIANCE_SERVICE");
        bytes32 depositKey = RateLimitHelpers.makeAssetDestinationKey(
            mainnetController.LIMIT_ASSET_TRANSFER(), address(usdc), address(buidlDeposit)
        );

        bytes32 redeemKey = mainnetController.LIMIT_BUIDL_REDEEM_CIRCLE();

        vm.startPrank(Ethereum.SPARK_PROXY);
        rateLimits.setRateLimitData(depositKey, 2_000_000e6, uint256(2_000_000e6) / 1 days);
        rateLimits.setRateLimitData(redeemKey, 2_000_000e6, uint256(2_000_000e6) / 1 days);
        vm.stopPrank();

        deal(address(usdc), address(almProxy), 2_000_000e6);

        // Step 1: Deposit into BUIDL
        vm.prank(relayer);
        mainnetController.transferAsset(address(usdc), buidlDeposit, 1_000_000e6);

        // Step 2: BUIDL gets minted into proxy
        assertEq(buidl.balanceOf(address(almProxy)), 0);

        vm.prank(admin);
        buidl.issueTokens(address(almProxy), 1_000_000e6);

        assertEq(buidl.balanceOf(address(almProxy)), 1_000_000e6);

        // Step 3: Malicious relayer spams transfers & redemptions
        // every iteration uses a cold `sload` so need at most 30e6 / 2100 = 14_286
        // the iterations gas cost is roughly linear per iteration, to speed up the test we scale down both
        ↪  iterations and gas used
        for (uint256 i; i < 10_000 / speedup; i++) {
            vm.prank(relayer);
            mainnetController.transferAsset(address(usdc), buidlDeposit, 1e6);
            vm.prank(admin);
            buidl.issueTokens(address(almProxy), 1e6);
        }

        // skip time lock
        skip(24 hours);
    }

    // run test in its own transaction so the sloads are cold
    function test_issuanceDos() public {
        // Step 4: Redeem non-malicious BUIDL after timelock is passed
        vm.startPrank(relayer);
        vm.expectRevert("SafeERC20: low-level call failed");
        mainnetController.redeemBUIDLCircleFacility{gas: 30e6 / speedup}(1_000_000e6);
        vm.stopPrank();
    }
}
```

MakerDAO: Acknowledged. The BUIDL team has stated that they use batching operationally to avoid this.

Cantina Managed: Acknowledged.

3.2 Low Risk

3.2.1 Temporary Metamorpho disruption by malicious relayer

Severity: Low Risk

Context: MainnetController.sol#L595-L626

Description: A malicious relayer with the curator role for a Metamorpho vault can perform the following attacks:

1. `setSupplyQueueMorpho([])`: By setting an empty supply queue, no new deposits can be performed. Temporary deposit DoS.
2. `reallocateMorpho()`: The relayer can reallocate all funds among the underlying Morpho Blue vaults (following the supply caps) to temporarily reduce the interest earned.

Recommendation: Once the malicious relayer is removed, the proper supply and withdrawal queue order can be restored.

MakerDAO: Acknowledged. We can add an attack vector test to demonstrate how this situation would be handled.

Cantina Managed: Acknowledged.

3.3 Informational

3.3.1 Missing `cancelDepositRequest`/`cancelRedeemRequest` from the Centrifuge integration

Severity: Informational

Context: MainnetController.sol#L363-L423

Description: In ERC7540, deposit and redeem operations are **not atomic**; executing either requires three steps:

1. Request.
2. Fulfill.
3. Claim.

The Centrifuge implementation introduces an additional feature: requests can be canceled so they can't be fulfilled. However, in `MainnetController`, only the request/claim functionality has been implemented, while the cancel feature is missing. This means the `almProxy` must always successfully complete any request, with no option to cancel.

```
/**
 * @dev Submits a Request for cancelling the pending deposit Request
 *
 * - controller MUST be msg.sender unless some unspecified explicit approval is given by the caller,
 *    approval of ERC-20 tokens from controller to sender is NOT enough.
 * - MUST set pendingCancelDepositRequest to `true` for the returned requestId after request
 * - MUST increase claimableCancelDepositRequest for the returned requestId after fulfillment
 * - SHOULD be claimable using `claimCancelDepositRequest`
 * Note: while `pendingCancelDepositRequest` is `true`, `requestDeposit` cannot be called
 */
function cancelDepositRequest(uint256 requestId, address controller) external;

/**
 * @dev Submits a Request for cancelling the pending redeem Request
 *
 * - controller MUST be msg.sender unless some unspecified explicit approval is given by the caller,
 *    approval of ERC-20 tokens from controller to sender is NOT enough.
 * - MUST set pendingCancelRedeemRequest to `true` for the returned requestId after request
 * - MUST increase claimableCancelRedeemRequest for the returned requestId after fulfillment
 * - SHOULD be claimable using `claimCancelRedeemRequest`
 * Note: while `pendingCancelRedeemRequest` is `true`, `requestRedeem` cannot be called
 */
function cancelRedeemRequest(uint256 requestId, address controller) external;
```

Recommendation: Implement the missing `cancelDepositRequest` and `cancelRedeemRequest` functions, allowing requests to be canceled. If these functions won't be implemented, consider documenting this. If these functions are implemented, the following additional claim functions should also be added:

- `claimCancelDepositRequest`.
- `claimCancelRedeemRequest`.

These functions would allow users to claim a canceled request.

MakerDAO: Fixed in PR 16.

Cantina Managed: Verified

3.3.2 Centrifuge integration can be temporarily DoSed if membership expires

Severity: Informational

Context: MainnetController.sol#L363-L423

Description: Every transfer in Centrifuge is guarded by a hook: `RestrictionManager.checkERC20Transfer(...)`. This function checks whether both the `from` and `to` addresses are frozen, whitelisted, or endorsed by `ROOT`.

- Centrifuge RestrictionManager:

```
// --- ERC1404 implementation ---
/// @inheritdoc IHook
function checkERC20Transfer(address from, address to, uint256, /* value */ HookData calldata hookData)
    public
    view
    returns (bool)
{
    if (uint128(hookData.from).getBit(FREEZE_BIT) == true && !root.endorsed(from)) {
        // Source is frozen and not endorsed
        return false;
    }

    if (root.endorsed(to) || to == address(0)) {
        // Destination is endorsed, and source was already checked, so the transfer is allowed
        return true;
    }

    uint128 toHookData = uint128(hookData.to);
    if (toHookData.getBit(FREEZE_BIT) == true) {
        // Destination is frozen
        return false;
    }

    if (toHookData >> 64 < block.timestamp) {
        // Destination is not a member
        return false;
    }

    return true;
}
```

Centrifuge whitelists an address in two ways:

1. By creating a membership with an expiration date.
2. By endorsing the address in `ROOT`.

Depending on how the `almProxy` is whitelisted, a DoS scenario may be possible:

- If the `almProxy` is whitelisted using a membership with an expiration date, it could be temporarily DoSed if the membership expires and Centrifuge does not renew it in time.

Recommendation: Coordinate with the Centrifuge team to whitelist the `almProxy` using one of the following approaches:

- Set the maximum membership expiration time: `type(uint64).max`.
- Directly endorse `almProxy` in `ROOT`, ensuring it is permanently whitelisted and not affected by membership expiration.

MakerDAO: Acknowledged. We will use the first option with Centrifuge team (whitelisting to `type(uint64).max`).

Cantina Managed: Acknowledged.

3.3.3 Malicious relayer can delay `requestMapleRedemption`

Severity: Informational

Context: MainnetController.sol#L576

Description: There can only ever be a single active withdrawal request per alm proxy in Maple's Withdrawal Manager.

```
function addShares(uint256 shares_, address owner_) external override onlyPoolManager {
    require(shares_ > 0,              "WM:AS:ZERO_SHARES");
    require(requestIds[owner_] == 0, "WM:AS:IN_QUEUE");

    // ...
}
```

A malicious relayer can call `requestMapleRedemption` with 1 share and no new request can be triggered until the existing request has been processed.

Recommendation: Once the malicious relayer is removed, one can cancel the previous request via `cancelMapleRedemption` to remove the pending request.

MakerDAO: Acknowledged. We can add an attack vector test to demonstrate how this issue would be resolved.

Cantina Managed: Acknowledged.

3.3.4 `requestRedeemERC7540` rate limit can be inaccurate

Severity: Informational

Context: MainnetController.sol#L401

Description: The actual `assets` received later in `claimRedeemERC7540` might differ from the `convertToAssets(shares)` used for the rate limit. This can happen if the redemption price changes.

```
/// @inheritdoc IERC7575
/// @notice     The calculation is based on the token price from the most recent epoch retrieved from
↪  Centrifuge.
///             The actual conversion MAY change between order submission and execution.
function convertToAssets(uint256 shares) public view returns (uint256 assets) {
    assets = manager.convertToAssets(address(this), shares);
}
```

If the rate-limite asset computation is underestimating the assets that can be redeemed later, the relayers can end up withdrawing more assets than the rate limit allowed.

Recommendation: At the point of requesting the redemption, the actual redemption price is yet unknown and no accurate estimation can be done.

MakerDAO: Acknowledged, we will document this behaviour.

Cantina Managed: Acknowledged.

3.3.5 Full source for BUIDL not available

Severity: Informational

Context: NA. **Description:** The BUIDL redemption process involves a sequence of checks performed by the `canTransfer` modifier of the `BUIDL` token. It calls into several services which are proxies with unverified `target` source code. It's unclear what logic runs in the called functions. For example, `0x07A1EBFb9a9A421249DDC71Bddb8860cc077E3a9` is the compliance service contract proxy that points to an unverified `target`.

Recommendation: We recommend that the BUIDL team verifies their deployed service contracts.

```
uint256 public constant TRUST_SERVICE = 1;
uint256 public constant DS_TOKEN = 2;
uint256 public constant REGISTRY_SERVICE = 4;
uint256 public constant COMPLIANCE_SERVICE = 8;
uint256 public constant UNUSED_1 = 16;
uint256 public constant WALLET_MANAGER = 32;
uint256 public constant LOCK_MANAGER = 64;
uint256 public constant PARTITIONS_MANAGER = 128;
uint256 public constant COMPLIANCE_CONFIGURATION_SERVICE = 256;
uint256 public constant TOKEN_ISSUER = 512;
uint256 public constant WALLET_REGISTRAR = 1024;
uint256 public constant OMNIBUS_TBE_CONTROLLER = 2048;
uint256 public constant TRANSACTION_RELAYER = 4096;
uint256 public constant TOKEN_REALLOCATOR = 8192;
uint256 public constant SECURITIZE_SWAP = 16384;
```

MakerDAO: Acknowledged.

Cantina Managed: Acknowledged.