SUPREMACY

Smart Contract
Security Audit Report

Prepared for R2 Protocol

Prepared by Supremacy

September 19, 2025

Contents

1 Introduction

Given the opportunity to review the design document and related codebase of the R2 Protocol, we outline in the report our systematic approach to evaluate potential security issues in the smart contract(s) implementation, and provide additional suggestions or recommendations for improvement. Our results show that the given version of smart contracts can be further improved due to the presence of several issues related to either security or performance. This document outlines our audit results.

1.1 About Client

R2 Protocol is building the first on-chain Co-designed Fund of Funds (FoF) — a yield platform that works directly with institutional partners to create customized RWA vaults tailored for crypto users.

Item	Description
Client	R2 Protocol
Type	Smart Contract
Languages	Solidity
Platform	EVM-compatible

1.2 Audit Scope

In the following, we show the Git repository of reviewed file and the commit hash used in this security audit:

- Repository: https://github.com/R2Yield/protocol-core/tree/main/src
- Commit Hash: 7a819e389a41e727254d0fb1f785357db17ee9e0

1.3 Changelogs

Version	Date	Description
0.1	September 15, 2025	Initial Draft
0.2	September 17, 2025	Release Candidate #1
1.0	September 19, 2025	Final Release

1.4 About Us

Supremacy is a leading blockchain security firm, composed of industry hackers and academic researchers, provide top-notch security solutions through our technology accumulation and innovative research.

We are reachable at X (https://x.com/SupremacyHQ), or Email (contact@supremacy.email).

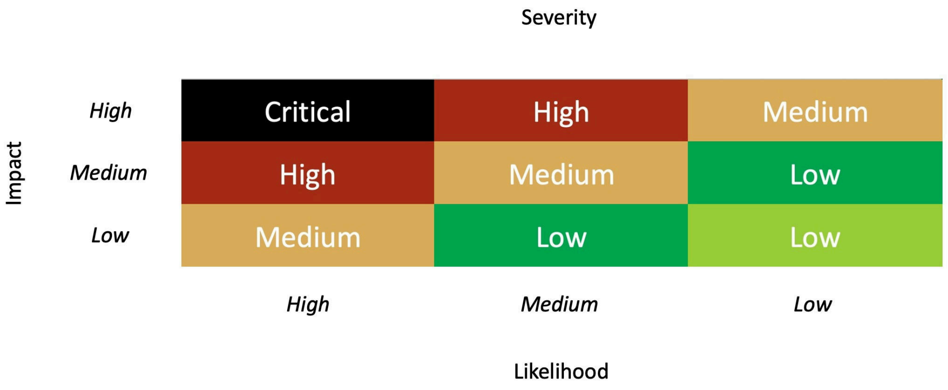
1.5 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- Likelihood represents the likelihood of a finding to be triggered or exploited in practice
- Impact specifies the technical and business-related consequences of a finding

- Severity is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.



Severity

Impact	*High*	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low
		High	*Medium*	*Low*

Likelihood

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

2 Findings

The table below summarizes the findings of the audit, including status and severity details.

ID	Severity	Description	Status
1	Critical	Permanent share loss on cancellation	Fixed
2	Medium	Potential arbitrage opportunities	Acknowledged
3	Medium	Potential denial of service	Fixed
4	Low	Lack of address verification	Fixed
5	Low	Lack of `safeTransfer`	Fixed
6	Informational	Unlock time can be bypassed	Acknowledged
7	Informational	Potential `currentShareTokenTvl` manipulation	Acknowledged
8	Informational	Centralization risk	Acknowledged
9	Informational	Redundant code	Acknowledged
10	Informational	Lack of comment	Acknowledged

2.1 Critical

1. Permanent share loss on cancellation [Critical]

`Severity: Critical` `Likelihood: High` `Impact: High`

`Status: Fixed`

Description

In the async redemption flow, `requestRedeem` escrows shares via `_update(owner, address(this), shares)`. On cancellation via `cancelRequestRedeem`, the function calls `_withdraw(address(this), receiver, address(this), assets, 0)`, which transfers `R2USD` but burns zero shares. The escrowed shares remain locked in the contract, permanently reducing the user's balance.

```
    function requestRedeem(
        uint256 shares,
        address receiver,
        address owner,
        uint256 referral
    ) external nonReentrant whenSetRequestManager virtual returns (uint256 requestId) {
        if (_msgSender() != owner) {
            _spendAllowance(owner, _msgSender(), shares);
        }
        uint256 maxShares = balanceOf(owner);
        if (shares > maxShares) {
            revert ERC4626ExceededMaxRedeem(owner, shares, maxShares);
        }
        uint256 assets = previewRedeem(shares);
        _update(owner, address(this), shares);
        requestId = IR2RequestManager(requestManager)
            .asyncRequestRedeem(owner, receiver, shares, assets, referral);
    }
```

R2YieldShareToken.sol

```
    function cancelRequestRedeem(uint256 requestId) external nonReentrant whenSetRequestManager virtual returns (uint256) {
        (uint8 state, address owner, address receiver, , uint256 assets) = IR2RequestManager(requestManager)
            .redeemRequestState(address(this), requestId);
        if (state != uint8(RequestState.Pending)) {
            revert InvalidRequestState(state);
        }
        if (msg.sender != owner && msg.sender != receiver) {
            revert InvalidRequestCaller(msg.sender);
        }
        IR2RequestManager(requestManager).cancelRequestRedeem(requestId);
        _withdraw(address(this), receiver, address(this), assets, 0);
        return assets;
    }
```

R2YieldShareToken.sol

Recommendation

Revise the code logic accordingly.

2.2 Medium

2. Potential arbitrage opportunities [Medium]

Severity: Medium Likelihood: Medium Impact: Medium

Status: Acknowledged

Description

The conversion function _convertToShares and _convertToAssets hard-codes a 1:1 pegging mechanism, disregarding potential slippage or de-pegging risks. Should R2USD experience de-pegging during operations, arbitrage opportunities may arise.

Recommendation

Revise the code logic accordingly.

3. Potential denial of service [Medium]

Severity: Medium Likelihood: Medium Impact: Medium

Status: Fixed

Description

The receive function in R2Minting uses .transfer to send msg.value to the owner. Since .transfer is subject to gas limits, if the owner is a multi-sig wallet, the transaction may revert due to insufficient gas.

```
    receive() external payable {
        emit Received(msg.sender, msg.value);
        payable(owner()).transfer(msg.value);
    }
```

R2Minting.sol

Recommendation

Consider using owner.call{value: msg.value}.

2.3 Low

4. Lack of address verification [Low]

Severity: Low Likelihood: Low Impact: Low

Status: Fixed

Description

The R2AssetVault contract's constructor, withdrawFor, and forceWithdraw functions lack zero address verification.

Recommendation

Revise the code logic accordingly.

5. Lack of safeTransfer [Low]

```
Severity: Low                    Likelihood: Low                    Impact: Low
Status: Fixed
```

Description

Failure to use `safeTransfer` may result in unexpected behavior for tokens that do not comply with EIP-20.

```
    function recoverTokens(address[] memory tokens, uint256[] memory amounts) public onlyOwner {
        require(tokens.length == amounts.length, "invalid");
        for (uint256 i; i < tokens.length; i++) {
            IERC20(tokens[i]).transfer(owner(), amounts[i]);
            emit RecoverToken(tokens[i], amounts[i]);
        }
    }
```

BaseAccessManager.sol

Recommendation

Revise the code logic accordingly.

2.4 Informational

6. Unlock time can be bypassed [Informational]

```
Status: Acknowledged
```

Description

If the `BaseAccessManager` has not been `initialized`, the unlock time can be bypassed.

```
    function transferMinter(address newMinter) public onlyOwner {
        emit MinterTransferStarted(minter, newMinter);
        pendingMinterUnlockTime = block.timestamp + 1 days;
        pendingMinter = newMinter;
    }

    function acceptMinter() public onlyOwner {
        require(pendingMinter != address(0), "No pending");
        if (initialized) {
            require(block.timestamp >= pendingMinterUnlockTime, "Minter transfer is still locked");
        }
        emit MinterTransferred(minter, pendingMinter);
        minter = pendingMinter;
        pendingMinter = address(0);
        pendingMinterUnlockTime = 0;
    }

    function transferRequestManager(address newRequestManager) public onlyOwner {
        emit RequestManagerTransferStarted(requestManager, newRequestManager);
        pendingRequestManagerUnlockTime = block.timestamp + 1 days;
        pendingRequestManager = newRequestManager;
    }

    function acceptRequestManager() public onlyOwner {
        require(pendingRequestManager != address(0), "No pending");
        if (initialized) {
            require(block.timestamp >= pendingRequestManagerUnlockTime, "RequestManager transfer is still locked");
        }
        emit RequestManagerTransferred(requestManager, pendingRequestManager);
        requestManager = pendingRequestManager;
        pendingRequestManager = address(0);
        pendingRequestManagerUnlockTime = 0;
    }
```

BaseAccessManager.sol

Recommendation

Revise the code logic accordingly.

7. Potential `currentShareTokenTvl` manipulation [Informational]

`Status: Acknowledged`

Description

The `requestDeposit` function in the R`2YieldRequestManager` contract relies on `IR2USD(IR2YieldShareToken(shareToken).asset()).balanceOf(shareToken)` to calculate `currentShareTokenTvl`. However, `balanceOf` is vulnerable to manipulation, which could lead to unintended outcomes if exploited by malicious actors.

Recommendation

Revise the code logic accordingly.

8. Centralization risk [Informational]

`Status: Acknowledged`

Description

In the R2 Protocol, there is a privilege account, which has the right to directly transfer a specific asset in the liquidity pool.

Our analysis shows that privileged accounts need to be scrutinized. In the following, we will examine privileged accounts and the associated privileged access in the current contract.

Note that if the privileged owner account is a plain EOA, this may be worrisome and pose counter-party risk to the protocol users. A multi-sig account could greatly alleviate this concern, though it is still far from perfect. Specifically, a better approach is to eliminate the administration key concern by transferring the role to a communitygoverned DAO. In the meantime, a timelock-based mechanism can also be considered as mitigation.

```
    function recoverTokens(address[] memory tokens, uint256[] memory amounts) public onlyOwner {
        require(tokens.length == amounts.length, "invalid");
        for (uint256 i; i < tokens.length; i++) {
            IERC20(tokens[i]).safeTransfer(owner(), amounts[i]);
            emit RecoverToken(tokens[i], amounts[i]);
        }
    }
```

BaseAccessManager.sol

Recommendation

Initially onboarding could can use multisign wallets or timelocks to initially mitigate centralization risks, but as a long-running protocol, we recommend eventually transfer the privileged account to the intended DAO-like governance contract. All changed to privileged operations may need to be mediated with necessary timelocks.

Eventually, activate the normal on-chain community-based governance life-cycle and ensure the intended trustless nature and high-quality distributed governance.

9. Redundant code [Informational]

`Status: Acknowledged`

Description

The `deposit`, `mint`, `redeem`, and `withdraw` functions of the `R2YieldShareToken` contract do not require the addition of a separate `referral` parameter. The redundant code can be removed.

Recommendation

Revise the code logic accordingly.

10. Lack of comment [Informational]

`Status: Acknowledged`

Description

Throughout the codebase there are numerous functions missing or lacking documentation. This hinders reviewers' understanding of the code's intention, which is fundamental to correctly assess not only security, but also correctness. Additionally, comments improve readability and ease maintenance. They should explicitly explain the purpose or intention of the functions, the scenarios under which they can fail, the roles allowed to call them, the values returned and the events emitted.

Recommendation

Consider thoroughly documenting all functions (and their parameters) that are part of the smart contracts' public interfaces. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing comments, consider following the Ethereum Natural Specification Format (NatSpec).

3 Disclaimer

This security audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. This security audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues, also cannot make guarantees about any additional code added to the assessed project after the audit version. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contract(s). Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.