BLOCKSEC

Security Audit
Report for Composer Contracts

Date: November 3, 2026 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Chapter 1 Introduction **1**

1.1 About the Audit Target . . . 1
1.2 Disclaimer . . . 2
1.3 Procedure of Auditing . . . 2
1.3.1 Security Issues . . . 2
1.3.2 Additional Recommendation . . . 3
1.4 Security Model . . . 3

Chapter 2 Findings **5**

2.1 Security Issue . . . 5
2.1.1 Potential DoS due to the donation attack in the contract `EmAssetForfeitComposer` 5
2.1.2 Lack of rate limit checks when initiating cooldown requests . . . 6
2.2 Recommendation . . . 8
2.2.1 Use `Ownable2Step` for ownership transfer in the contract `EmAssetForfeitComposer` 8
2.2.2 Add slippage protections in the contract `EmAssetVaultComposer` . . . 8
2.3 Note . . . 11
2.3.1 Potential centralization risks . . . 11
2.3.2 Properly deploy OFT adapter contracts . . . 11
2.3.3 Set peers correctly for each `OApp` contract . . . 11

Report Manifest

Item	Description
Client	Inverter Network
Target	Composer Contracts

Version History

Version	Date	Description
1.0	November 3, 2026	First release

Signature



Digitally signed by: BlockSec (https://blocksec.com)

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2026-11-04 09:40:52+0000

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 14 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About the Audit Target

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The audit target (hereinafter referred to as the Target) of this audit is the code repository [1] of Composer Contracts of Inverter Network.

Composer Contracts of Inverter Network implement a LayerZero-based staking and bridging system for emUSD, refactored from the earlier iTry cross-chain architecture. Users deposit emUSD into the wemUSD vault in exchange for yield-bearing shares and can exit through cooldown-based unstaking or fast redemption. The system supports both hub-local and cross-chain deposits, redemptions, and unstake claims: emUSD is bridged through a burn-and-mint OFT, while wemUSD is transferred through a canonical-share adapter. It additionally provides a cross-chain retirement flow that burns wemUSD shares and confirms the corresponding emUSD burn for the retirement of the relevant RWA position.

Note this audit only focuses on the codes located in the following directories/files.

- crosschain/shares/StakedEmAssetOFT.sol
- crosschain/composer/EmAssetVaultComposer.sol
- crosschain/shares/StakedEmAssetOFTAdapter.sol
- crosschain/composer/libraries/VaultComposerSync.sol
- crosschain/composer/EmAssetUnstakeMessenger.sol
- crosschain/composer/interfaces/IEmAssetVaultComposer.sol
- crosschain/composer/interfaces/IEmAssetUnstakeMessenger.sol
- crosschain/composer/libraries/IVaultComposerSync.sol
- crosschain/composer/StakedEmAssetInstantRedeemForwarder.sol
- crosschain/bridge/EmAssetOFTBridge.sol
- crosschain/retirement/EmAssetForfeitComposer.sol
- crosschain/retirement/EmAssetForfeitReceiver.sol

Other files are not within the scope of the audit. Additionally, all dependencies of the Target are considered reliable in terms of both functionality and security, and are therefore not included in the audit scope.

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report. Code prior to and including the baseline version (`Version 0`), where applicable, is outside the scope of this audit and assumes to be reliable and secure.

[1] âĂćhttps://github.com/InverterNetwork/emusd-monorepo

Project	Version	Commit Hash
emusd-monorepo	Version 1	4518a705fb701f8abd28fb8ed601eae4b392e1a9
	Version 2	a37f0d07aa3e714ed11f5ea45cdc163895ed7413

1.2 Disclaimer

This 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 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 Target, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of the Target.

The scope of this audit is limited to the code mentioned in Section 1.1. 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.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan the Target with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of the Target and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Security Issues

* Access control
* Permission management
* Whitelist and blacklist mechanisms
* Initialization consistency
* Improper use of the proxy system
* Reentrancy
* Denial of Service (DoS)
* Untrusted external call and control flow

- Exception handling
- Data handling and flow
- Events operation
- Error-prone randomness
- Oracle security
- Business logic correctness
- Semantic and functional consistency
- Emergency mechanism
- Economic and incentive impact

1.3.2 Additional Recommendation

- Gas optimization
- Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [2] and Common Weakness Enumeration [3]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Low*
High	High	Medium
Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[3] https://cwe.mitre.org/

Furthermore, the status of a discovered item will fall into one of the following five categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Partially Fixed** The item has been confirmed and partially fixed by the client.
- **Fixed** The item has been confirmed and fixed by the client.

Chapter 2 Findings

In total, we found **two** potential security issues. Besides, we have **two** recommendations and **three** notes.

- Low Risk: 2
- Recommendation: 2
- Note: 3

ID	Severity	Description	Category	Status
1	Low	Potential DoS due to the donation attack in the contract `EmAssetForfeitComposer`	Security Issue	Fixed
2	Low	Lack of rate limit checks when initiating cooldown requests	Security Issue	Confirmed
3	-	Use `Ownable2Step` for ownership transfer in the contract `EmAssetForfeitComposer`	Recommendation	Fixed
4	-	Add slippage protections in the contract `EmAssetVaultComposer`	Recommendation	Fixed
5	-	Potential centralization risks	Note	-
6	-	Properly deploy OFT adapter contracts	Note	-
7	-	Set peers correctly for each `OApp` contract	Note	-

The details are provided in the following sections.

2.1 Security Issue

2.1.1 Potential DoS due to the donation attack in the contract `EmAssetForfeitComposer`

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `EmAssetForfeitComposer`, the function `lzCompose()` handles forfeit requests by invoking the function `forfeit()` of the contract `ShakedEmAssetFastRedeem`. When a full forfeit request is received, the function `_forfeit()` in the contract `ShakedEmAssetFastRedeem` performs a check `assets == maximumAssets` to ensure all assets will be burned. However, this design may be vulnerable to a donation attack. Specifically, a malicious user could donate 1 wei asset into the vault to break the check. As a result, the full forfeit request may suffer a DoS issue due to the donation attack.

```
128   function lzCompose(address from, bytes32 guid, bytes calldata message, address, bytes calldata
          )
129     external
130     payable
131     override
132   {
133     if (msg.sender != address(endpoint)) revert OnlyEndpoint(msg.sender);
```

```
134     if (from != shareOft) revert OnlyShareOft(from);
135     uint256 shares = message.amountLD();
136     // A restricted recipient receives no credit. Complete its compose without spending held
            shares.
137     if (shares == 0) return;
138
139     bytes memory payload = message.composeMsg();
140     uint256 assetsToBurn;
141     uint256 callbackRequested;
142     if (payload.length == 32) {
143       assetsToBurn = abi.decode(payload, (uint256));
144     } else if (payload.length == 64) {
145       (assetsToBurn, callbackRequested) = abi.decode(payload, (uint256, uint256));
146       if (callbackRequested > 1) revert InvalidPayload();
147     } else {
148       revert InvalidPayload();
149     }
150     if (callbackRequested == 0 && msg.value != 0) revert UnexpectedNativeValue();
151     if (assetsToBurn == 0) revert InvalidAmount();
152     vault.forfeit(shares, assetsToBurn);
```

Listing 2.1: emusd-monorepo/packages/foundry/contracts/crosschain/retirement/EmAssetForfeitComposer.sol

```
126 function _forfeit(uint256 amount, uint256 assets) private {
127     uint256 maximumAssets = previewRedeem(amount);
128     if (assets > maximumAssets) revert ExcessiveForfeitAssets();
129     if (amount == totalSupply() && assets != maximumAssets) revert FullForfeitAssetsMismatch();
130     _withdraw(msg.sender, address(this), msg.sender, assets, amount);
131     _burnVaultAssets(assets);
132
133     emit Forfeited(msg.sender, amount, assets);
134 }
```

Listing 2.2: emusd-monorepo/packages/foundry/contracts/staking/StakedEmAssetFastRedeem.sol

Impact The full forfeit request may suffer a DoS issue due to the donation attack.

Suggestion Revise the logic accordingly.

2.1.2 Lack of rate limit checks when initiating cooldown requests

Severity Low

Status Confirmed

Introduced by `Version 1`

Description In the contract `EmAssetVaultComposer`, the function `_initiateCooldown()` can handle multiple cooldown requests for the sender by refreshing their cooldown states (i.e., the variables `cooldownEnd` and `underlyingAmount`) via the function `_startComposerCooldown()`. Specifically, the cooldown amount of each request is accumulated in the variable `underlyingAmount`. Once the time `cooldownEnd` is reached, the sender is allowed to claim all accumulated assets

in one request. However, when accumulating the cooldown amount, the rate limit check is missing for the variable `underlyingAmount` in the function `_startComposerCooldown()`. As a result, when the accumulated cooldown amount exceeds the limit, the sender's assets become locked.

```
239   function _initiateCooldown(bytes32 _redeemer, uint256 _shareAmount) internal virtual {
240     address redeemer = _redeemer.bytes32ToAddress();
241     if (redeemer == address(0)) revert ZeroAddress();
242     _revertIfFullRestricted(redeemer);
243     uint256 assetAmount = IStakedEmAssetCrosschain(address(VAULT)).cooldownSharesByComposer(
            _shareAmount, redeemer);
244     emit CooldownInitiated(_redeemer, redeemer, _shareAmount, assetAmount);
245   }
```

Listing 2.3: emusd-monorepo/packages/foundry/contracts/crosschain/composer/EmAssetVaultComposer.sol

```
221     function _startComposerCooldown(address composer, address redeemer, uint256 shares, uint256
            assets) private {
222         _revertIfRestricted(redeemer);
223
224         uint104 cooldownEnd = uint104(block.timestamp) + cooldownDuration;
225
226         // Effects: record cooldown state first
227         cooldowns[redeemer].cooldownEnd = cooldownEnd;
228         cooldowns[redeemer].underlyingAmount += uint152(assets);
229
230         // Interaction: external token transfer to the silo (protected by nonReentrant modifier)
231         _withdraw(composer, address(silo), composer, assets, shares);
232
233         emit ComposerCooldownInitiated(composer, redeemer, shares, assets, cooldownEnd);
234     }
235 }
```

Listing 2.4: emusd-monorepo/packages/foundry/contracts/staking/StakedEmAssetCrosschain.sol

```
336   function _send(address _oft, SendParam memory _sendParam, address _refundAddress) internal {
337     if (_sendParam.dstEid == VAULT_EID) {
338       address erc20 = _oft == ASSET_OFT ? ASSET_ERC20 : SHARE_ERC20;
339
340       if (msg.value > 0) revert NoMsgValueExpected();
341       IERC20(erc20).safeTransfer(_sendParam.to.bytes32ToAddress(), _sendParam.amountLD);
342     } else {
343       (,, uint192 limit,) = IRateLimitedOFT(_oft).rateLimits(_sendParam.dstEid);
344       uint256 rate = IOFTDecimalConversion(_oft).decimalConversionRate();
345       uint256 amountLD = _sendParam.amountLD - _sendParam.amountLD % rate;
346       if (limit != 0 && amountLD > limit) revert AmountExceedsRateLimit(amountLD, limit);
347
348       uint256 fee = IOFT(_oft).quoteSend(_sendParam, false).nativeFee;
349       if (msg.value < fee) revert InsufficientMsgValue(fee, msg.value);
350       IOFT(_oft).send{value: msg.value}(_sendParam, MessagingFee(msg.value, 0), _refundAddress);
351     }
```

```
352     }
```

Listing 2.5: emusd-monorepo/packages/foundry/contracts/crosschain/composer/libraries/VaultComposerSync.sol

Impact Potential lock of funds due to the lack of rate limit checks when handling cooldown requests.

Suggestion Add the rate limit check in the function `_startComposerCooldown()`.

Feedback from the project The project stated that this is acceptable since the user can always withdraw on the hub side.

2.2 Recommendation

2.2.1 Use `Ownable2Step` for ownership transfer in the contract `EmAssetForfeitComposer`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description Compared to other contracts (e.g., `EmAssetVaultComposer`), the contract `EmAssetForfeitComposer` does not use the 2-step ownership transfer mechanism. It is recommended to use the contract `Ownable2Step` for the ownership transfer in the contract `EmAssetForfeitComposer`.

```
71  contract EmAssetVaultComposer is VaultComposerSync, IEmAssetVaultComposer, OApp, Ownable2Step {
```

Listing 2.6: emusd-monorepo/packages/foundry/contracts/crosschain/composer/EmAssetVaultComposer.sol

```
39  contract EmAssetForfeitComposer is ILayerZeroComposer, OAppSender, OAppOptionsType3,
        ReentrancyGuard {
40    using OFTComposeMsgCodec for bytes;
41    using SafeERC20 for IERC20;
```

Listing 2.7: emusd-monorepo/packages/foundry/contracts/crosschain/retirement/EmAssetForfeitComposer.sol

Suggestion Use the contract `Ownable2Step` for the ownership transfer in the contract `EmAssetForfeitComposer`.

2.2.2 Add slippage protections in the contract `EmAssetVaultComposer`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `EmAssetVaultComposer`, all redemption-related requests (i.e., fast redeem, cooldown, and unstake requests) lacks a slippage protection (i.e., via the function `_assertSlippage()`) compared to the deposit requests (i.e., via the function `_depositAndSend()`). It is recommended to add slippage protections accordingly.

```
258   function _fastRedeem(bytes32 _redeemer, uint256 _shareAmount, SendParam memory _sendParam,
          address _refundAddress)
259     internal
260     virtual
261   {
262     address redeemer = _redeemer.bytes32ToAddress();
263     if (redeemer == address(0)) revert ZeroAddress();
264
265     if (IAccessControl(ASSET_ERC20).hasRole(BLACKLISTED_ROLE, redeemer)) revert
            ReceiverBlacklisted();
266     _revertIfFullRestricted(redeemer);
267
268     (uint256 assets,) = instantRedeemForwarder.instantRedeem(_shareAmount, address(this));
269
270     if (assets == 0) {
271       revert NoAssetsToRedeem();
272     }
273
274     _sendParam.amountLD = assets;
275     // The source helper does not enforce a minimum after native OFT dust removal.
276     _sendParam.minAmountLD = 0;
277
278     _send(ASSET_OFT, _sendParam, _refundAddress);
279
280     emit CrosschainFastRedeemProcessed(redeemer, _sendParam.dstEid, _shareAmount, assets);
281   }
```

Listing 2.8: emusd-monorepo/packages/foundry/contracts/crosschain/composer/EmAssetVaultComposer.sol

```
239   function _initiateCooldown(bytes32 _redeemer, uint256 _shareAmount) internal virtual {
240     address redeemer = _redeemer.bytes32ToAddress();
241     if (redeemer == address(0)) revert ZeroAddress();
242     _revertIfFullRestricted(redeemer);
243     uint256 assetAmount = IStakedEmAssetCrosschain(address(VAULT)).cooldownSharesByComposer(
            _shareAmount, redeemer);
244     emit CooldownInitiated(_redeemer, redeemer, _shareAmount, assetAmount);
245   }
```

Listing 2.9: emusd-monorepo/packages/foundry/contracts/crosschain/composer/EmAssetVaultComposer.sol

```
357   function _handleUnstake(
358     Origin calldata _origin,
359     bytes32 _guid,
360     IEmAssetUnstakeMessenger.UnstakeMessage memory unstakeMsg
361   ) internal virtual {
362     address user = unstakeMsg.user;
363
364     if (user == address(0)) revert ZeroAddress();
365     if (_origin.srcEid == 0) revert InvalidOrigin();
366     if (!spokeClaimAllowed[user][_origin.srcEid]) revert SpokeClaimNotAllowed(user, _origin.
            srcEid);
```

```
367
368     // The destination token rules decide delivery, not the hub asset blacklist.
369     _revertIfFullRestricted(user);
370
371     if (msg.value < unstakeMsg.msgValue) {
372       revert InsufficientMsgValueForReturn(unstakeMsg.msgValue, msg.value);
373     }
374
375     uint256 assets = IStakedEmAssetCrosschain(address(VAULT)).unstakeThroughComposer(user);
376
377     // A cooldown below the OFT precision would clear for a zero payout.
378     uint256 rate = IOFTDecimalConversion(ASSET_OFT).decimalConversionRate();
379     if (assets - assets % rate == 0) {
380       revert NoAssetsToUnstake();
381     }
382
383     bytes memory options = OptionsBuilder.newOptions();
384
385     SendParam memory _sendParam = SendParam({
386       dstEid: _origin.srcEid,
387       to: bytes32(uint256(uint160(user))),
388       amountLD: assets,
389       minAmountLD: 0,
390       extraOptions: options,
391       composeMsg: "",
392       oftCmd: ""
393     });
394
395     _send(ASSET_OFT, _sendParam, address(this));
396
397     emit CrosschainUnstakeProcessed(user, _origin.srcEid, assets, _guid);
398   }
```

Listing 2.10: emusd-monorepo/packages/foundry/contracts/crosschain/composer/EmAssetVaultComposer.sol

```
225   function _depositAndSend(
226     bytes32 _depositor,
227     uint256 _assetAmount,
228     SendParam memory _sendParam,
229     address _refundAddress
230   ) internal virtual {
231     uint256 shareAmount = _deposit(_depositor, _assetAmount);
232     // A local transfer delivers all shares. A remote OFT send removes dust.
233     uint256 deliveredAmount = shareAmount;
234     if (_sendParam.dstEid != VAULT_EID) {
235       uint256 rate = IOFTDecimalConversion(SHARE_OFT).decimalConversionRate();
236       deliveredAmount -= deliveredAmount % rate;
237     }
238     _assertSlippage(deliveredAmount, _sendParam.minAmountLD);
```

Listing 2.11: emusd-monorepo/packages/foundry/contracts/crosschain/composer/libraries/VaultComposerSync.sol

Suggestion Add slippage protections in the contract `EmAssetVaultComposer`.

2.3 Note

2.3.1 Potential centralization risks

Introduced by

Description In this project, several privileged roles (e.g., `owner`) can conduct sensitive operations, which introduces potential centralization risks. For example, the `owner` could update peers for OFT contracts or rescue assets. If the private keys of the privileged accounts are lost or maliciously exploited, it could pose a significant risk to the protocol.

2.3.2 Properly deploy OFT adapter contracts

Introduced by `Version 1`

Description The omnichain architecture should be designed and operated under the assumption that only one adapter (e.g., the contracts `StakedEmAssetOFTAdapter` and `StakedEmAssetOFTAdapter` instance exists across the deployment. This constraint is essential to preserve unified liquidity accounting and to ensure that bridging operations have a consistent, globally correct source of supply. Furthermore, deploying multiple adapter instances can fragment liquidity across chains and create situations where outbound transfers are executed against an adapter that cannot guarantee sufficient destination side availability, which may lead to irrecoverable token loss when the destination adapter lacks adequate supply.

2.3.3 Set peers correctly for each `OApp` contract

Introduced by `Version 1`

Description To ensure reliable cross-chain messaging, peer relationships should be configured on both the source and destination chains by calling the function `setPeer()`. Both chains' `OApp` contracts are expected to perform peer verification prior to message execution on the destination chain, which depends on accurate, symmetric peer configuration. Furthermore, deployments and operational procedures should consistently validate that each `OApp` contract has its intended peer set for every supported remote chain.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS