Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.03.22, the SlowMist security team received the Open Social team's security audit application for Open Social Core, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project team should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Suggestion	There are better practices for coding or architecture.

2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

Serial Number	Audit Class	Audit Subclass
1	Overflow Audit	-
2	Reentrancy Attack Audit	-
3	Replay Attack Audit	-
4	Flashloan Attack Audit	-
5	Race Conditions Audit	Reordering Attack Audit
6	Permission Vulnerability Audit	Access Control Audit
		Excessive Authority Audit
7	Security Design Audit	External Module Safe Use Audit
		Compiler Version Security Audit
		Hard-coded Address Security Audit
		Fallback Function Safe Use Audit
		Show Coding Security Audit
		Function Return Value Security Audit
		External Call Function Security Audit

Serial Number	Audit Class	Audit Subclass
7	Security Design Audit	Block data Dependence Security Audit
		tx.origin Authentication Security Audit
8	Denial of Service Audit	-
9	Gas Optimization Audit	-
10	Design Logic Audit	-
11	Variable Coverage Vulnerability Audit	-
12	"False Top-up" Vulnerability Audit	-
13	Scoping and Declarations Audit	-
14	Malicious Event Log Audit	-
15	Arithmetic Accuracy Deviation Audit	-
16	Uninitialized Storage Pointer Audit	-

3 Project Overview

3.1 Project Introduction

Open Social is a decentralised social project that allows users to create and join communities and to post in the communities they join or interact with posts made by other users. (e.g. commenting, liking, following users, etc.) Joining a community and following other users will receive a corresponding nft.

3.2 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

NO	Title	Category	Level	Status
N1	Issue with reusable	Design Logic Audit	High	Fixed

NO	Title	Category	Level	Status
	joinNFT			
N2	Lack of the repeatability check for creating OpenReactions	Design Logic Audit	Medium	Ignored
N3	Missing inviter check when creating a profile	Design Logic Audit	Suggestion	Fixed
N4	Lack of check when following other users	Design Logic Audit	Suggestion	Fixed
N5	Lack of check whether a community has been joined	Design Logic Audit	Medium	Fixed
N6	Missing event record	Others	Suggestion	Fixed
N7	call() should be used instead of transfer() and send()	Gas Optimization Audit	Suggestion	Fixed
N8	Risk of excessive authority	Authority Control Vulnerability Audit	Medium	Acknowledged
N9	Authority transfer enhancement	Others	Suggestion	Fixed

4 Code Overview

4.1 Contracts Description

Audit Version:

https://gitlab.com/Keccak256-evg/opensocial/core

commit: 9fda72fe30220f0ad70809121b8f0ec205f4671b

Fixed Version:

https://github.com/Open-Social-Protocol/osp-core

commit: ea36dbb2bf0aa843a364a1b1fa2c371bbe2e8082

The main network address of the contract is as follows:

The code was not deployed to the mainnet.

4.2 Visibility Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

ActivityExtensionBase			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	OspContext
initializeActivityExtension	External	Payable	onlyOsp
_initializeActivityExtension	Internal	Can Modify State	-
supportsInterface	External	-	-

CommunityCondBase			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	OspContext
processCreateCommunity	External	Payable	onlyOsp
supportsInterface	External	-	-
_processCreateCommunity	Internal	Can Modify State	-

EIP712Base			
Function Name	Visibility	Mutability	Modifiers
_validateRecoveredAddress	Internal	-	-
_calculateDomainSeparator	Internal	-	-
_calculateDigest	Internal	-	-

ERC6551Account			
Function Name	Visibility	Mutability	Modifiers
onERC721Received	External	-	-
onERC1155Received	External	-	-
onERC1155BatchReceived	External	-	-
supportsInterface	External	-	-

JoinConditionBase			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	OspContext
supportsInterface	External	-	-
initializeCommunityJoinCondition	External	Can Modify State	onlyOsp
processJoin	External	Payable	onlyOsp
_initializeCommunityJoinCondition	Internal	Can Modify State	-
_processJoin	Internal	Can Modify State	-

OpenReactionBase			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	OspContext
processReaction	External	Payable	onlyOsp
_processReaction	Internal	Can Modify State	-
supportsInterface	External	-	-

OspContext			
Function Name	Visibility	Mutability	Modifiers

OspContext			
<Constructor>	Public	Can Modify State	-

OspMultiState			
Function Name	Visibility	Mutability	Modifiers
protocolStateStorage	Internal	-	-
_getState	Internal	-	-
_setState	Internal	Can Modify State	-
_validatePublishingEnabled	Internal	-	-
_validateNotPaused	Internal	-	-

OspNFTBase			
Function Name	Visibility	Mutability	Modifiers
_initialize	Internal	Can Modify State	initializer
_update	Internal	Can Modify State	-
_afterTokenTransfer	Internal	Can Modify State	-
supportsInterface	Public	-	-
_increaseBalance	Internal	Can Modify State	-

OspSBTBase			
Function Name	Visibility	Mutability	Modifiers
_afterTokenTransfer	Internal	Can Modify State	-

ReferenceConditionBase			
Function Name	Visibility	Mutability	Modifiers

ReferenceConditionBase			
<Constructor>	Public	Can Modify State	OspContext
supportsInterface	External	-	-
initializeReferenceCondition	External	Can Modify State	onlyOsp
processReactionReference	External	Payable	onlyOsp
_initializeReferenceCondition	Internal	Can Modify State	-
_processReactionReference	Internal	Can Modify State	-

SlotNFTCommunityCond			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	CommunityCondBase
_processCreateCommunity	Internal	Can Modify State	nonPayable
whitelistCommunitySlot	External	Can Modify State	onlyGov
isCommunitySlotWhitelisted	External	-	-
isSlotNFTUsable	External	-	-
_validateSlotNFT	Internal	-	-

WhitelistAddressCommunityCond			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	CommunityCondBase
_processCreateCommunity	Internal	Can Modify State	nonPayable
setMaxCreationNumber	External	Can Modify State	onlyGov
allowedCreationNumber	External	-	-

ERC20FeeJoinCond			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	JoinConditionBase
_initializeCommunityJoinCondition	Internal	Can Modify State	-
_processJoin	Internal	Can Modify State	nonPayable
getCommunityData	External	-	-

HoldTokenJoinCond			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	JoinConditionBase
_initializeCommunityJoinCondition	Internal	Can Modify State	-
_processJoin	Internal	Can Modify State	nonPayable
getCommunityData	External	-	-

NativeFeeJoinCond			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	JoinConditionBase
_initializeCommunityJoinCondition	Internal	Can Modify State	onlyOsp
_processJoin	Internal	Can Modify State	onlyOsp
getCommunityData	External	-	-

OnlyMemberReferenceCond			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ReferenceConditionBase
_initializeReferenceCondition	Internal	Can Modify State	-

OnlyMemberReferenceCond			
_processReactionReference	Internal	Can Modify State	nonPayable

FeeConditionBase			
Function Name	Visibility	Mutability	Modifiers
_validateDataIsExpected	Internal	-	-
_tokenWhitelisted	Internal	-	-

CommunityLogic			
Function Name	Visibility	Mutability	Modifiers
createCommunity	External	Payable	whenNotPaused
emitCommunityNFTTransferEvent	External	Can Modify State	-
setJoinCondition	External	Can Modify State	whenNotPaused
getCommunityTokenURI	External	-	-
getCommunity	External	-	-
getJoinNFT	External	-	-
getJoinCondition	External	-	-
getCommunityIdByHandle	Public	-	-
getCommunityAccount	Public	-	-
getCommunityAccount	External	-	-
_setJoinCondition	Internal	Can Modify State	-
_createCommunity	Internal	Can Modify State	-
_initJoinCondition	Private	Can Modify State	-
_deployJoinNFT	Internal	Can Modify State	-

CommunityLogic			
_deployCommunityTBA	Internal	Can Modify State	-
_validateCommunityCondition	Internal	Can Modify State	-
_initHandle	Internal	Can Modify State	-
_validateCallerIsCommunityOwner	Internal	-	-
_communityNFT	Internal	-	-

ContentLogic			
Function Name	Visibility	Mutability	Modifiers
createActivity	External	Payable	whenPublishingEnabled
createActivityWithSig	External	Can Modify State	whenPublishingEnabled
createComment	External	Payable	whenPublishingEnabled
createCommentWithSig	External	Can Modify State	whenPublishingEnabled
createOpenReaction	External	Payable	whenPublishingEnabled
createOpenReactionWithSig	External	Can Modify State	whenPublishingEnabled
createMegaphone	External	Can Modify State	whenPublishingEnabled
getContentCount	External	-	-
getContent	External	-	-
getCommunityIdByContent	External	-	-
_createActivity	Internal	Can Modify State	-
_createComment	Internal	Can Modify State	-
_createMegaphone	Internal	Can Modify State	-
_createOpenReaction	Internal	Can Modify State	-
_initReferenceCondition	Internal	Can Modify State	-

ContentLogic			
_initActivityExtension	Internal	Can Modify State	-
_initOpenReaction	Internal	Can Modify State	-
_validateReferenced	Internal	Can Modify State	-
_validateCallerIsJoinCommunity	Internal	-	-

GovernanceLogic			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can Modify State	initializer
setState	External	Can Modify State	onlyRole
whitelistSuperCommunityCreator	External	Can Modify State	onlyRole
whitelistApp	External	Can Modify State	onlyRole
whitelistToken	External	Can Modify State	onlyRole
reserveCommunityHandle	External	Can Modify State	onlyRole
setBaseURI	External	Can Modify State	onlyRole
setERC6551AccountImpl	External	Can Modify State	onlyRole
updateMetadata	External	Can Modify State	onlyRole
setIdStart	External	Can Modify State	onlyRole
isSuperCommunityCreatorWhitelisted	External	-	-
isAppWhitelisted	External	-	-
isTokenWhitelisted	External	-	-
isReserveCommunityHandle	External	-	-
getFollowSBTImpl	External	-	-
getJoinNFTImpl	External	-	-

GovernanceLogic			
getCommunityNFT	External	-	-
getERC6551AccountImpl	External	-	-
getState	External	-	-
getBaseURI	External	-	-
eip712Domain	External	-	-

OspLogicBase			
Function Name	Visibility	Mutability	Modifiers
_checkFollowCondition	Internal	-	-
_checkCommunityCondition	Internal	-	-
_checkJoinCondition	Internal	-	-
_checkActivityExtension	Internal	-	-
_checkOpenReaction	Internal	-	-
_checkReferenceCondition	Internal	-	-
_validateIsProfileOwner	Internal	-	-
_validateHasProfile	Internal	-	-
_ownerOf	Internal	-	-
_calculateDomainSeparator	Internal	-	-

ProfileLogic			
Function Name	Visibility	Mutability	Modifiers
createProfile	External	Can Modify State	whenNotPaused
setFollowCondition	External	Can Modify State	whenNotPaused

ProfileLogic			
burn	External	-	whenNotPaused
safeTransferFrom	External	-	-
safeTransferFrom	External	-	-
transferFrom	External	-	-
approve	External	-	-
setApprovalForAll	External	-	-
supportsInterface	External	-	-
ownerOf	Public	-	-
name	External	-	-
symbol	External	-	-
tokenURI	External	-	-
balanceOf	Public	-	-
totalSupply	External	-	-
nonces	External	-	-
getFollowSBT	External	-	-
getFollowCondition	External	-	-
getHandle	External	-	-
getProfileIdByHandle	External	-	-
getProfileIdByAddress	External	-	-
getProfile	External	-	-
tokenOfOwnerByIndex	External	-	-
tokenByIndex	External	-	-

ProfileLogic			
getApproved	External	-	-
isApprovedForAll	External	-	-
_mint	Internal	Can Modify State	-
_validateHandle	Internal	-	-
_createProfile	Internal	Can Modify State	-
_setFollowCondition	Internal	Can Modify State	-
_addTokenToAllTokensEnumeration	Private	Can Modify State	-
_initFollowCondition	Private	Can Modify State	-

RelationLogic			
Function Name	Visibility	Mutability	Modifiers
follow	External	Payable	whenNotPaused
batchFollow	External	Payable	whenNotPaused
join	External	Payable	whenNotPaused
batchJoin	External	Payable	whenNotPaused
getFollowSBTURI	External	-	-
getJoinNFTURI	External	-	-
isFollow	External	-	-
isJoin	External	-	-
emitFollowSBTTransferEvent	External	Can Modify State	-
emitJoinNFTTransferEvent	External	Can Modify State	-
_follow	Internal	Can Modify State	-
_batchFollow	Internal	Can Modify State	-

RelationLogic			
_executeFollow	Internal	Can Modify State	-
_deployFollowSBT	Internal	Can Modify State	-
_join	Internal	Can Modify State	-
_batchJoin	Internal	Can Modify State	-
_executeJoin	Internal	Can Modify State	-

LikeReaction			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	OpenReactionBase
_processReaction	Internal	-	-

VoteReaction			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	OpenReactionBase
_processReaction	Internal	-	-

OspStorage			
Function Name	Visibility	Mutability	Modifiers
_getProfileStorage	Internal	-	-
_getContentStorage	Internal	-	-
_getGovernanceStorage	Internal	-	-
_getCommunityStorage	Internal	-	-

CommunityNFT			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	-
mint	External	Can Modify State	-
tokenURI	Public	-	-
_afterTokenTransfer	Internal	Can Modify State	-
updateMetadata	External	Can Modify State	-
setIdStart	External	Can Modify State	-

FollowSBT			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	-
mint	External	Can Modify State	-
tokenURI	Public	-	-
_afterTokenTransfer	Internal	Can Modify State	-

JoinNFT			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	-
mint	External	Can Modify State	-
getSourceCommunityPointer	External	-	-

JoinNFT			
tokenURI	Public	-	-
_afterTokenTransfer	Internal	Can Modify State	-

CommunityAccountProxy			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
_implementation	Internal	-	-

FollowSBTProxy			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
_implementation	Internal	-	-

JoinNFTProxy			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
_implementation	Internal	-	-

OspRouterImmutable			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
routerStorage	Internal	-	-
supportsInterface	Public	-	-
changeAdmin	Public	Can Modify State	onlyAdmin

OspRouterImmutable			
addRouter	External	Can Modify State	onlyAdmin
updateRouter	External	Can Modify State	onlyAdmin
removeRouter	External	Can Modify State	onlyAdmin
getRouterForFunction	Public	-	-
getAllFunctionsOfRouter	External	-	-
getAllRouters	External	-	-
_implementation	Internal	-	-
_getRouterForFunction	Public	-	-
_addRouter	Internal	Can Modify State	-
_updateRouter	Internal	Can Modify State	-
_removeRouter	Internal	Can Modify State	-
_changeAdmin	Internal	Can Modify State	-

OspUniversalProxy			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	OspContext ERC1967Proxy
updateToAndCall	External	Can Modify State	onlyGov

4.3 Vulnerability Summary

[N1] [High] Issue with reusable joinNFT

Category: Design Logic Audit

Content

In the RelationLogic contract, users can join a specified community by calling the join or batchJoin function. A joinNFT is minted for the users, and the processJoin function in the community-specific JoinCondition contract is

executed to enforce the corresponding join conditions.

However, the joinNFT transfers are not restricted in any way, which leads to the following scenario:

If a user joins a community that requires a fee, and pays a specified fee to obtain a joinNFT for the community, he can then transfer that NFT to other users, and since the transfer of the joinNFT is not subject to any checking, multiple users can view or manipulate the community's resources or information using a single NFT (paying for it only once).

Code Location: contracts/core/logic/RelationLogic.sol

```
    function _executeJoin(
        uint256 communityId,
        bytes calldata joinConditionData,
        uint256 value
    ) internal returns (uint256 tokenId) {
        OspDataTypes.CommunityStruct memory community =
_getCommunityStorage()._communityById[
            communityId
        ];
        if (community.joinNFT == address(0)) revert OspErrors.InvalidCommunityId();
        tokenId = IJoinNFT(community.joinNFT).mint(msg.sender);
        if (community.joinCondition != address(0)) {
            IJoinCondition(community.joinCondition).processJoin{value: value}(
                msg.sender,
                communityId,
                joinConditionData
            );
        }
    }
```

Solution

It is recommended to set joinNFT as a non-transferable attribute or perform relevant checking and join condition execution operations on the from and to addresses according to the corresponding JoinCondition address when transferring. (Judge according to the actual business requirements)

Status

Fixed

[N2] [Medium] Lack of the repeatability check for creating OpenReactions

Category: Design Logic Audit

Content

In the ContentLogic contract, users can interact with posts by calling the createOpenReaction function, for example by liking a post. However there is no check in the _createOpenReaction function to see if a user has already interacted with a post. This could result in a user being able to repeatedly like or top a post.

Code Location: contracts/core/logic/ContentLogic.sol

```
    function _createOpenReaction(OspDataTypes.CreateOpenReactionData calldata vars)
internal {
        _validateCallerIsJoinCommunity(vars.profileId, vars.communityId);
        if (msg.value < vars.reactionValue) {
            revert OspErrors.InvalidValue();
        }
        _validateReferenced(
            vars.profileId,
            vars.communityId,
            vars.referencedProfileId,
            vars.referencedContentId,
            vars.referenceConditionData,
            msg.value - vars.reactionValue
        );

        (address openReaction, bytes memory openReactionData) = _initOpenReaction(
            vars.profileId,
            vars.referencedProfileId,
            vars.referencedContentId,
            vars.reactionAndData,
            vars.reactionValue
        );

        emit OspEvents.OpenReactionCreated(
            vars.profileId,
            vars.referencedProfileId,
            vars.referencedContentId,
            vars.communityId,
            openReaction,
            openReactionData,
            vars.ctx,
            block.timestamp
        );
    }
```

Solution

It is recommended that the status of a user's interaction on a post should be checked to avoid double liking or topping a post.

Status

Ignored; Response from the project team: The OpenReaction contract does not record and check the state, which is checked when the interaction takes place down the chain.

[N3] [Suggestion] Missing inviter check when creating a profile

Category: Design Logic Audit

Content

In the ProfileLogic contract, the user can create a profile by calling the createProfile function, but here the inviter parameter is not checked. If the inviter passed in is equal to msg.sender, it does not follow the normal logic.

Code Location: contracts/core/logic/ProfileLogic.sol#L274

```
    function _createProfile(
        OspDataTypes.CreateProfileData calldata vars
    ) internal returns (uint256) {
        ...

        if (vars.inviter != 0) {
            if (_getProfileStorage()._profileById[vars.inviter].owner == address(0))
{
                revert OspErrors.ProfileDoesNotExist();
            }
            profileStruct.inviter = vars.inviter;
        }

        ...
    }
```

Solution

It is recommended that the incoming inviter parameter be checked.

Status

Fixed

[N4] [Suggestion] Lack of check when following other users

Category: Design Logic Audit

Content

In the RelationLogic contract, anyone can follow other users by calling the follow or batchFollow function. But here it doesn't check if the object to follow is equal to msg.sender, which doesn't follow the normal logic if you can follow yourself.

Code Location: contracts/core/logic/RelationLogic.sol#L186-211

```
    function _executeFollow(
        uint256 profileId,
        bytes calldata followConditionData,
        uint256 value
    ) internal returns (uint256 tokenId) {
        if (_getProfileStorage()._profileById[profileId].owner == address(0))
            revert OspErrors.TokenDoesNotExist();

        address followCondition =
_getProfileStorage()._profileById[profileId].followCondition;
        address followSBT = _getProfileStorage()._profileById[profileId].followSBT;

        if (followSBT == address(0)) {
            followSBT = _deployFollowSBT(profileId);
            _getProfileStorage()._profileById[profileId].followSBT = followSBT;
        }

        ...
    }
```

Solution

It is recommended to check in the _executeJoin function that the following user cannot be equal to msg.sender.

Status

Fixed

[N5] [Medium] Lack of check whether a community has been joined

Category: Design Logic Audit

Content

In the RelationLogic contract, anyone can join a specified community by calling the join or batchJoin function. However, there is no check to see if the user has already joined the community. If the condition of joining the

community is that the number of tokens held reaches a set value, then the user can join the community several times to mint joinNFT, and then transfer the NFT to other users (even if the other users' token balances don't meet the requirements).

Code Location: contracts/core/logic/RelationLogic.sol#L318-335

```
    function _executeJoin(
        uint256 communityId,
        bytes calldata joinConditionData,
        uint256 value
    ) internal returns (uint256 tokenId) {
        OspDataTypes.CommunityStruct memory community =
_getCommunityStorage()._communityById[
            communityId
        ];
        if (community.joinNFT == address(0)) revert OspErrors.InvalidCommunityId();
        tokenId = IJoinNFT(community.joinNFT).mint(msg.sender);

        ...
    }
```

Solution

It is recommended to check if the user currently calling the function has already joined the community.

Status

Fixed

[N6] [Suggestion] Missing event record

Category: Others

Content

The following functions in several contracts are for event logging of key parameter settings.

Code Location:

contracts/core/conditions/community/SlotNFTCommunityCond.sol#L34-36

```
    function whitelistCommunitySlot(address slot, bool whitelist) external
onlyOperation {
        _slotNFTWhitelisted[slot] = whitelist;
    }
```

contracts/core/conditions/community/WhitelistAddressCommunityCond.sol#L37-39

```
    function setMaxCreationNumber(address to, uint256 _maxCreationNumber) external
onlyOperation {
        maxCreationNumber[to] = _maxCreationNumber;
    }
```

Code Location: contracts/core/logic/GovernanceLogic.sol#L108-112

```
    function setERC6551AccountImpl(
        address accountImpl
    ) external override onlyRole(Constants.GOVERNANCE) {
        _getGovernanceStorage()._erc6551AccountImpl = accountImpl;
    }
```

Solution

It is recommended to record events when sensitive parameters are modified for self-inspection or community review.

Status

Fixed

[N7] [Suggestion] call() should be used instead of transfer() and send()

Category: Gas Optimization Audit

Content

The transfer() and send() functions forward a fixed amount of 2300 gas. Historically, it has often been recommended to use these functions for value transfers to guard against reentrancy attacks. However, the gas cost of EVM instructions may change significantly during hard forks which may break already deployed contract systems that make fixed assumptions about gas costs. For example. EIP 1884 broke several existing smart contracts due to a cost increase of the SLOAD instruction.

Code location:

contracts/core/conditions/join/NativeFeeJoinCond.sol#L51

```
    function _processJoin(
        address follower,
```

```
        uint256 communityId,
        bytes calldata data
    ) internal override onlyOsp {
        ...

        payable(_dataByCommunity[communityId].recipient).transfer(value);
        (follower, data); //unused
    }
```

Solution

It is recommended to use call() instead of transfer(), but be sure to respect the CEI pattern or add re-entrancy guards, and the return value should be checked.

Status

Fixed

[N8] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

In the OspUniversalProxy contract, the gov role can directly upgrade the implementation contract and call the functions of the new contract. If the privilege is lost or misused, This could lead to malicious tampering with the contract's functionality.

Code location:

contracts/upgradeability/OspUniversalProxy.sol

```
    function updateToAndCall(address newImplementation, bytes memory data) external
onlyGov {
        ERC1967Utils.upgradeToAndCall(newImplementation, data);
    }
```

Solution

It is recommended that in the early stages of the project, the core role should use multi-signatures and the time-lock contract to avoid single-point risks. After the project is running stably, the authority of these roles should be handed over to community governance for management, and strict identity authentication should be performed when adding roles. If certain roles are run via automated scripts (e.g. keeper or Liquidator, then the permissions

on that account should be tightly controlled, with private keys and helpers kept in isolation).

Status

Acknowledged; Response from the project team: Safe multi-signature wallet management will be used after the project goes live.

[N9] [Suggestion] Authority transfer enhancement

Category: Others

Content

In the Account contract, the admin role does not adopt the pending and access processes. If the admin is incorrectly set, the permission will be lost.

Code Location:

contracts/upgradeability/OspRouterImmutable.sol

```
    function changeAdmin(address _admin) public onlyAdmin {
        _changeAdmin(_admin);
    }

    ...

    function _changeAdmin(address admin_) internal {
        Data storage data = routerStorage();
        emit AdminChanged(data.admin, admin_);
        data.admin = admin_;
    }
```

Solution

It is recommended to adopt the pending and access processes. Only the new pauserMultisig role can accept the permissions to transfer.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002404100002	SlowMist Security Team	2024.03.22 - 2024.04.10	Medium Risk

Summary conclusion: The SlowMist security team uses a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 1 high risk, 3 medium risk, 5 suggestion vulnerabilities. And 1 medium risk vulnerability was ignored; 1 medium risk vulnerability was acknowledged; All other findings were fixed. Since the code has not yet been deployed to the main network and the permissions of the core roles have not been transferred and managed, the reported risk level is medium risk for the time being.

6 Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist