Cyberscope

Audit Report

Tea-Fi

April 2025

Files PermitManager.sol, Permitable.sol., PermitManagement.sol

Audited by © cyberscope

Table of Contents

Review

Audit Updates

Initial Audit	09 Apr 2025 https://github.com/cyberscope-io/audits/blob/main/1-tea/v1/permitManager.pdf
Corrected Phase 2	28 Apr 2025

Source Files

Filename	SHA256
Permitable.sol	19303a4ca1b1a5348aec81f0af7ac60eec9c255d9fe803a31127949392ec45f4
PermitManager.sol	19303a4ca1b1a5348aec81f0af7ac60eec9c255d9fe803a31127949392ec45f4
PermitManagement.sol	c30673e9347e789530a8bea9c423e2df94af430d49eaa656adddb1131dcb7539

Overview

PermitManager Contract

The `PermitManager` contract is responsible for managing token transfer approvals via on-chain permits. It executes permit-based token transfers across multiple standards, significantly reducing the number of signatures needed for transactions.

Purpose and Functionality

The core purpose of `PermitManager` is to offload the complexity of token approvals and ensure seamless interactions with whitelisted contracts. It supports multiple permit formats, including:

- EIP-2612
- DAI-style permits
- Permit2

The contract uses a generalized internal mechanism (`_tryPermit`) to detect the permit format based on the signature length and structure, and then dispatches the correct call using low-level assembly logic.

Role-Based Access Control

To maintain a secure environment, the contract uses OpenZeppelin's `AccessControl` to define two key roles:

- `DEFAULT_ADMIN_ROLE` : Typically assigned to the Tea-Fi multisig wallet, it has the authority to manage spenders.
- `SPENDER_ROLE` : Assigned to contracts or addresses that are allowed to call `executePermitTransfer` and `executePermitTransferBatch` .

Only accounts with `SPENDER_ROLE` can initiate token transfers using permits, ensuring a secure and auditable transaction flow.

Token Transfers via Permits

The main public functions `executePermitTransfer` and `executePermitTransferBatch` allow authorized roles to perform one or multiple token transfers by utilizing existing token allowances and permits.

If the required allowance does not exist, the contract attempts to generate it via a permit. If no valid permit is provided or if it fails to execute, the transaction reverts.

Whitelisting and Management

The contract allows the admin to dynamically manage spenders:

- **Adding spenders**: Grants the `SPENDER_ROLE` to new addresses.
- **Removing spenders**: Revokes the `SPENDER_ROLE` from specified addresses.

Security and Extensibility

The `PermitManager` contract is constructed with best practices in mind:

- Only trusted roles may execute transfers. Trusted roles inherit the `PermitManagement` abstract contract to ensure that payments are handled appropriately.
- All external interactions are validated to avoid misuse or zero-address vulnerabilities.
- Signature parsing is handled securely and efficiently using inline assembly for gas optimization and compatibility across standards.

Roles

PermitManager Contract

Admins

Administrators with the `DEFAULT_ADMIN_ROLE` (typically the Tea-Fi multisig wallet) can interact with the following functions:

- `function addSpenders(address[] calldata spenders)`
- `function removeSpenders(address[] calldata spenders)`

Spenders

Whitelisted contracts or addresses with the `SPENDER_ROLE` can perform token transfers using on-chain permits:

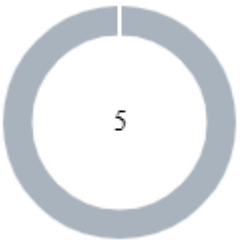
- `function executePermitTransfer(PermitTransferParams calldata params)`
- `function executePermitTransferBatch(PermitTransferParams[] calldata params)`

Internal Logic

While not directly exposed to users, the following internal functions enable safe and compliant execution of permit logic:

- `_makeTokenPermit(address token, address owner, bytes calldata permit)`
- `_makePermit2(address token, address owner, uint256 amount, bytes calldata permit2Data)`
- `_transferPayment(address token, address owner, address to, uint256 amount)`
- `_safePermit(IERC20 token, address owner, bytes calldata permit)`
- `_tryPermit(IERC20 token, address owner, address spender, bytes calldata permit)`

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	5

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	1	4	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	CCR	Contract Centralization Risk	Acknowledged
	PPII	Potential Permit Implementation Inconsistency	Acknowledged
	SRR	Spender Role Risk	Acknowledged
	L17	Usage of Solidity Assembly	Acknowledged
	L20	Succeeded Transfer Check	Unresolved

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	PermitManager.sol#L71,85
Status	Acknowledged

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

```
function addSpenders(address[] calldata spenders) external
onlyRole(DEFAULT_ADMIN_ROLE)

function removeSpenders(address[] calldata spenders) external
onlyRole(DEFAULT_ADMIN_ROLE)
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

Team Update

The team has acknowledged that this is not a security issue and states:*The DEFAULT_ADMIN_ROLE is intentionally designed to be decentralized and not concentrated within a single entity. It is assigned to a secure multi-signature wallet that requires multiple approvals (e.g., 2 out of 3 or 3 out of 5 confi rmations) for executing actions. This approach distributes decision-making authority and mitigates risks associated with a single point of failure.*

PPII - Potential Permit Implementation Inconsistency

Criticality	Minor / Informative
Location	Permitable.sol#L36
Status	Acknowledged

Description

`_makeTokenPermit` depends on the token's implementation for gasless approvals. In case the implementation does not check chain specific information or if the nonce is not correctly updated, it is possible that an off-chain signature can be used in multiple chains or multiple times in the same chain.

```
function _makeTokenPermit(address token, address owner, bytes
calldata permit) internal {
    if (IERC20(token).allowance(owner, address(permit2)) ==
type(uint256).max) return;
    _safePermit(IERC20(token), owner, permit);
}
```

Recommendation

The team should carefully manage which token implementations are allowed to be used by the permit manager. Additionally, the team could consider validating the chain and the nonce in the permit manager to avoid the possibility of signature reusage.

Team Update

The team has acknowledged that this is not a security issue and states:*Tea-Fi operates only with whitelisted and audited tokens.*

SRR - Spender Role Risk

Criticality	Minor / Informative
Location	PermitManager.sol#L58
Status	Acknowledged

Description

The security of the system depends on the trustworthiness and correctness of the spender. If spender fails to ensure that the owner field in the `PermitTransferParams` matches the actual transaction sender (`msg.sender`), a malicious user could submit a valid permit signed by another user and unauthorizedly trigger a token transfer using someone else's approval. Without strong validation within the spender contract, this could allow signature replay or permit misuse.

```
function executePermitTransfer(PermitTransferParams calldata
params) public onlyRole(SPENDER_ROLE) {
    if (params.owner == address(0) || params.recipient ==
address(0) || params.token == address(0)) {
        revert ZeroAddress();
    }

    if (params.tokenData.length > 0)
_makeTokenPermit(params.token, params.owner, params.tokenData);

    if (params.permit2Data.length > 0)
_makePermit2(params.token, params.owner, params.amount,
params.permit2Data);

    _transferPayment(params.token, params.owner,
params.recipient, params.amount);
}
```

Recommendation

It is recommended that `SPENDER_ROLE` is only granted to contracts that ensure the owner field in `PermitTransferParams` matches the actual caller. This ensures that only the user who signed the permit can trigger its usage, preventing malicious actors from submitting permits signed by others.

Team Update

The team has acknowledged that this is not a security issue and states: *In order to resolve this issue, we introduced an abstract PermitManagement smart-contract that will be required to be implemented by every SPENDER_ROLE smart-contract.*

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	Permitable.sol#L103
Status	Acknowledged

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly ("memory-safe") {
            // solhint-disable-line no-inline-assembly
            let ptr := mload(0x40)

            // Switch case for different permit lengths,
indicating different permit standards
            switch permit.length
...
            }
            // Unknown
            default {
                mstore(ptr, permitLengthError)
                revert(ptr, 4)
            }
        }
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

Team Update

The team has acknowledged that this is not a security issue and states: *The private* `_tryPermit` *function, sourced from 1inch's AggregationRouterV6 (* `0x111111125421cA6dc452d289314280a0f8842A65` *), uses assembly for efficient and secure permit validation.*

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	PermitManagement.sol#L41
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
IERC20(token).transferFrom(owner, recipient, amount)
```

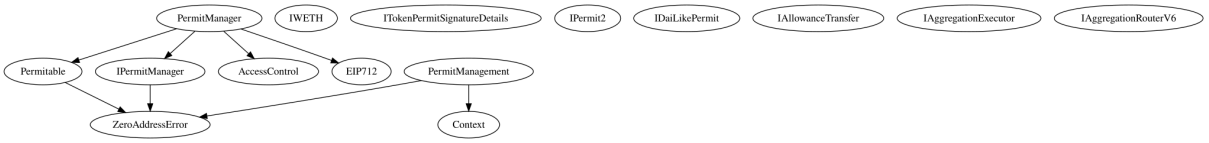
Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

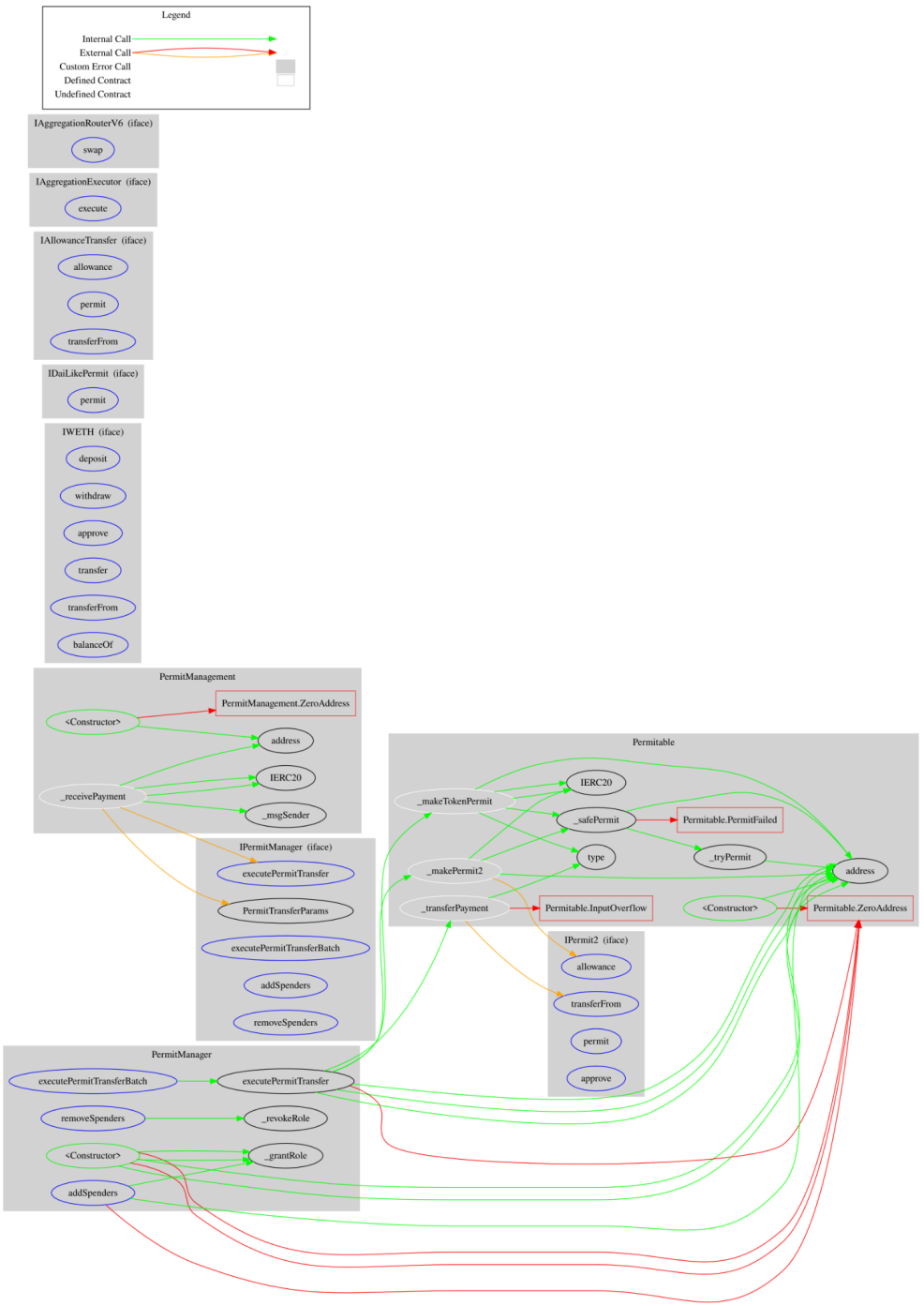
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Permitable	Implementation	ZeroAddress Error		
		Public	✓	-
	_makeTokenPermit	Internal	✓	
	_makePermit2	Internal	✓	
	_transferPayment	Internal	✓	
	_safePermit	Private	✓	
	_tryPermit	Private	✓	
PermitManager	Implementation	Permitable, IPermitManager, AccessControl, EIP712		
		Public	✓	Permitable EIP712
	executePermitTransferBatch	External	✓	onlyRole
	executePermitTransfer	Public	✓	onlyRole
	addSpenders	External	✓	onlyRole
	removeSpenders	External	✓	onlyRole
PermitManagement	Implementation	ZeroAddress Error, Context		
		Public	✓	-
	_receivePayment	Internal	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
IAggregationRouterV6 (iface)
swap
IAggregationExecutor (iface)
execute
IAllowanceTransfer (iface)
allowance
permit
transferFrom
IDaiLikePermit (iface)
permit
IWETH (iface)
deposit
withdraw
approve
transfer
transferFrom
balanceOf
PermitManagement
PermitManagement.ZeroAddress
<Constructor>
address
IERC20
_receivePayment
_msgSender
IPermitManager (iface)
executePermitTransfer
PermitTransferParams
executePermitTransferBatch
addSpenders
removeSpenders
Permitable
IERC20
_makeTokenPermit
_safePermit
Permitable.PermitFailed
type
_tryPermit
address
_makePermit2
_transferPayment
Permitable.InputOverflow
<Constructor>
Permitable.ZeroAddress
IPermit2 (iface)
allowance
transferFrom
permit
approve
PermitManager
executePermitTransferBatch
executePermitTransfer
removeSpenders
_revokeRole
<Constructor>
_grantRole
addSpenders

Summary

Tea-Fi contract implements a permit utility mechanism. This audit investigates security issues, business logic concerns and potential improvements.

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
Critical	Highly Likely / High Impact
Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
Minor / Informative	Unlikely / Low to no Impact

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io