ParaSwap PortikusV1

Security Audit Report

August 2, 2024

Contents

1 Introduction

1.1 About ParaSwap PortikusV1

`PortikusV1` is an intent-based protocol that allows agents to execute gasless swaps for signed user intents. It employs the Diamond Proxy architecture (`EIP-2535`) to enable future upgradability through adding various facets that facilitate different types of order execution. The protocol utilizes an executor-agent architecture where the `ExecutorManager` contract manages permissions, allowing certain agents to settle orders through specific partner executor contracts.

1.2 Audit Scope

This is the repository and commit id we used in the audit:

- https://github.com/paraswap/portikus-contracts/tree/main
- CommitID: 369ff46

And this is the final version representing all fixes implemented for the issues identified in the audit:

- https://github.com/paraswap/portikus-contracts/tree/main
- CommitID: f387460

Note that this audit covers all the smart contracts in the `src` folder excluding `src/diamond/*` and `executors/example/ExampleExecutor.sol`.

2 | Overall Assessment

This report has been compiled to identify issues and vulnerabilities within the `ParaSwap PortikusV1` protocol. Throughout this audit, we identified several issues spanning various severity levels. By employing auxiliary tool techniques to supplement our thorough manual code review, we have discovered the following findings.

Severity	Count	Acknowledged	Won't Do	Addressed
Critical	-	-	-	-
High	1	-	-	1
Medium	1	1	-	-
Low	2	-	-	2
Informational	-	-	-	-
Total	4	1	-	3

3 | Vulnerability Summary

3.1 Overview

Click on an issue to jump to it, or scroll down to see them all.

~~H-1~~ Untrusted External Call in ParaswapDelta

M-1 Potential Risks Associated with Centralization

~~L-1~~ Revisited Errors Used in SignatureLib::verify()

~~L-2~~ Lack of Interface to Remove Agent

3.2 Security Level Reference

In web3 smart contract audits, vulnerabilities are typically classified into different severity levels based on the potential impact they can have on the security and functionality of the contract. Here are the definitions for critical-severity, high-severity, medium-severity, and low-severity vulnerabilities:

Severity	Description
C-X (Critical)	A severe security flaw with immediate and significant negative consequences. It poses high risks, such as unauthorized access, financial losses, or complete disruption of functionality. Requires immediate attention and remediation.
H-X (High)	Significant security issues that can lead to substantial risks. Although not as severe as critical vulnerabilities, they can still result in unauthorized access, manipulation of contract state, or financial losses. Prompt remediation is necessary.
M-X (Medium)	Moderately impactful security weaknesses that require attention and remediation. They may lead to limited unauthorized access, minor financial losses, or potential disruptions to functionality.
L-X (Low)	Minor security issues with limited impact. While they may not pose significant risks, it is still recommended to address them to maintain a robust and secure smart contract.
I-X (Informational)	Warnings and things to keep in mind when operating the protocol. No immediate action required.
U-X (Undetermined)	Identified security flaw requiring further investigation. Severity and impact need to be determined. Additional assessment and analysis are necessary.

3.3 Vulnerability Details

[H-1] Untrusted External Call in ParaswapDelta

Target	Category	IMPACT	LIKELIHOOD	STATUS
ParaswapDelta.sol	Input Validation	High	Medium	Addressed

The `Portikus` protocol utilizes an executor-agent architecture where the `ExecutorManager` contract manages permissions, allowing certain agents to settle orders through specific partner executor contracts. Specifically, the `ParaswapDelta` contract is ParaSwap's own executor contract, responsible for handling the actual swap logic. While examining the execution of the swap operation in `ParaswapDelta`, we notice that a malicious agent can fake an `addAuthorizedAgent()` call as the swap call to bypass the ownership check and authorize any agent to `ParaswapDelta`.

In the following, we show the code snippet of the `ParaswapDelta::_executeCalldataAndTransferFees()` function, which is invoked by `Portikus` to complete the swap operation. Specifically, it parses the executor data (line 206), executes the swap on the execution address with the provided calldata (line 208), and transfers the fee to the fee recipient if needed (line 214). Note that the executor data, i.e., `executorData`, is provided by the agent. If the executor data provided by a malicious agent gets executed in `ParaswapDelta`, it may introduce unexpected impacts to the protocol.

For example, a malicious agent can craft a call to `PORTIKUS_V1.updateAgentAuthorization(...)` in `ParaswapDelta`, which can bypass the ownership check in the `addAuthorizedAgent()` function (line 63) and add any agent to the authorized list of `ParaswapDelta`.

Our study recommends implementing robust validation for the input executor data to prevent the execution of untrusted external calls.

ParaswapDelta::_executeCalldataAndTransferFees()

```
199 function _executeCalldataAndTransferFees(bytes calldata executorData)
200   internal
201   virtual
202   override
203   returns (bool success)
204 {
205   // Parse the executor data
206   ExecutorData memory data = abi.decode(executorData, (ExecutorData));
207   // Execute the swap on the execution address with the provided calldata
208   (bool executionSuccess,) = data.executionAddress.call(data.calldataToExecute);
209   // Return false if the execution failed
210   if (!executionSuccess) {
211       return false;
212   }
213   // Transfer the fee to the fee recipient if needed and ETH if the dest token
```

```
        is ETH
214     PORTIKUS_V1.transferFeesAndETH(data.feeRecipient, data.destToken, data.
            feeAmount);
215     // Return true if the execution was successful
216     return true;
217 }
```

PortikusV1PartnerExecutor::addAuthorizedAgent()

```
63 function addAuthorizedAgent(address agent) external virtual onlyOwner {
64   IExecutorManager(PORTIKUS_V1).updateAgentAuthorization(agent, true);
65 }
```

Remediation Add proper validation for the input executor data to ensure the integrity and trustworthiness of the executor data before executing it.

[M-1] Potential Risks Associated with Centralization

Target	Category	IMPACT	LIKELIHOOD	STATUS
Multiple Contracts	Security	Medium	Low	Acknowledged

In the `ParaSwap PortikusV1` protocol, the existence of the privileged `owner` account introduces centralization risks, as it holds significant control and authority over critical operations governing the protocol. In the following, we show the representative functions potentially affected by the privileges associated with the privileged `owner` account.

Example Privileged Operations in `ExecutorManager`

```
18 function updateExecutorAuthorization(address _executor, bool authorized)
       external onlyOwner {
19     // Check if the executor already has the authorized agents
20     if (executorAgentsList[_executor].length != 0) {
21         // If the executor is being unauthorized, remove all authorized agents
22         if (!authorized) {...}
23     } else {
24         // Update the executor authorization status
25         executors[_executor] = authorized;
26     }
27     emit ExecutorAuthorizationUpdated(_executor, authorized);
28 }
```

Example Privileged Operations in `PortikusV1PartnerExecutor`

```
63 function addAuthorizedAgent(address agent) external virtual onlyOwner {
64     IExecutorManager(PORTIKUS_V1).updateAgentAuthorization(agent, true);
65 }
```

Remediation To mitigate the identified issue, it is recommended to introduce multi-sig mechanism to undertake the role of the privileged account. Moreover, it is advisable to implement timelocks to govern all modifications to the privileged operations.

Response By Team We will use a timelock/multisig as owner to mitigate centralisation risks.

[L-1] Revisited Errors Used in SignatureLib::verify()

Target	Category	IMPACT	LIKELIHOOD	STATUS
SignatureLib.sol	Coding Practices	Low	N/A	Addressed

In the `SignatureLib` library, the `verify()` function is used to verify the signature for the hash and signer. If the verification fails, whether due to an invalid signature or an invalid signer, it reports respective errors. However, our study shows that it applies wrong errors for the failures. Specifically, if the verification fails due to an invalid signature (line 47), it should report `InvalidSignature()`, and if the verification fails due to invalid signer (line 48), it should report `InvalidSigner()`.

SignatureLib::verify()

```
37 function verify(bytes memory signature, bytes32 hash, address signer) internal
       view {
38     // Check if signer is an EOA
39     // - If signer is an EOA, check EIP-2098 signature format
40     // - If signer is a contract, check ERC-1271 signature format
41     if (signer.code.length == 0) {
42         if (signature.length == 64) {
43             (bytes32 r, bytes32 vs) = abi.decode(signature, (bytes32, bytes32));
44             bytes32 s = vs & UPPER_BIT_MASK;
45             uint8 v = uint8(uint256(vs >> 255)) + 27;
46             address actualSigner = ecrecover(hash, v, r, s);
47             if (actualSigner == address(0)) revert InvalidSigner();
48             if (actualSigner != signer) revert InvalidSignature();
49         } else {
50             revert InvalidSignature();
51         }
52     }
53     ...
54 }
```

Remediation Revisit the `verify()` function and report the correct errors for different failures.

[L-2] Lack of Interface to Remove Agent

Target	Category	IMPACT	LIKELIHOOD	STATUS
PortikusV1PartnerExecutor.sol	Business Logic	Low	N/A	Addressed

The `Portikus` protocol utilizes an executor-agent architecture which allows the authorized agents to settle orders through specific partner executor contracts. An agent is authorized by invoking the `ExecutorManager::updateAgentAuthorization()` by the executor contract owner. However, while examining the management of agents in the `PortikusV1PartnerExecutor` contract, we notice that it only supports adding an agent into the authorized list, without an interface to remove an authorized agent.

To facilitate the management of agents in the `PortikusV1PartnerExecutor` contract, it is suggested to add support for removing an authorized agent from the executor. This will enhance the control and security of the system by enabling the removal of agents who are no longer trusted or needed.

PortikusV1PartnerExecutor::addAuthorizedAgent()

```
40 function addAuthorizedAgent(address agent) external virtual onlyOwner {
41     IExecutorManager(PORTIKUS_V1).updateAgentAuthorization(agent, true);
42 }
```

Remediation Revisit the implementation of the `PortikusV1PartnerExecutor` contract to support the removal of an authorized agent.

4 Appendix

4.1 About AstraSec

AstraSec is a blockchain security company that serves to provide high-quality auditing services for blockchain-based protocols. With a team of blockchain specialists, AstraSec maintains a strong commitment to excellence and client satisfaction. The audit team members have extensive audit experience for various famous DeFi projects. AstraSec's comprehensive approach and deep blockchain understanding make it a trusted partner for the clients.

4.2 Disclaimer

The information provided in this audit report is for reference only and does not constitute any legal, financial, or investment advice. Any views, suggestions, or conclusions in the audit report are based on the limited information and conditions obtained during the audit process and may be subject to unknown risks and uncertainties. While we make every effort to ensure the accuracy and completeness of the audit report, we are not responsible for any errors or omissions in the report.

We recommend users to carefully consider the information in the audit report based on their own independent judgment and professional advice before making any decisions. We are not responsible for the consequences of the use of the audit report, including but not limited to any losses or damages resulting from reliance on the audit report.

This audit report is for reference only and should not be considered a substitute for legal documents or contracts.

4.3 Contact

Phone	+86 176 2267 4194
Email	contact@astrasec.ai
Twitter	https://twitter.com/AstraSecAI