MiloTruck

Optimism

DeputyPauseModule

January 7, 2025

Contents

1 Introduction

1.1 About MiloTruck

MiloTruck is an independent security researcher and 1/4th of the team at Renascence Labs. Currently, he works as a Lead Security Researcher at Spearbit and Lead Auditor at Trust Security.

For private audits or security consulting, please reach out on Twitter @milotruck.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an 'as-is' and 'as-available' basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	High	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

1.3.1 Impact

- High - Funds are **directly** at risk, or a **severe** disruption of the protocol's core functionality
- Medium - Funds are **indirectly** at risk, or **some** disruption of the protocol's functionality
- Low - Funds are **not** at risk

1.3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

2 Executive Summary

2.1 About Optimism

Optimism is an EVM equivalent Layer 2 blockchain connected to Ethereum. The OP Stack is the standardized, shared, and open-source development stack that makes it easy to spin up your own production-ready Layer 2 blockchain just like OP Mainnet. The Superchain is a network of OP Stack chains that share a bridging protocol, governance system, and more.

2.2 Overview

Project	Optimism
Repository	optimism
Commit Hash	2f17e6b67c61...
Mitigation Hash	4851c96c25e1...
Date	20 December 2024 - 21 December 2024

2.3 Issues Found

Severity	**Count**
High Risk	0
Medium Risk	0
Low Risk	3
Informational	2
Total Issues	**5**

3 Findings Summary

ID	Description	Status
L-1	Error from low-level call is not bubbled up correctly	Acknowledged
L-2	`deputyGuardianModule` must not be set to the `FOUNDATION_SAFE` address	Resolved
L-3	Previous signatures can still be used even when `deputyGuardianModule` is changed	Acknowledged
I-1	`execTransactionFromModuleReturnData()` does not fail when `deputyGuardianModule` is an address with no code	Resolved
I-2	Design choices regarding `pause()`	Acknowledged

4 Findings

Low Risk

[L-1] Error from low-level call is not bubbled up correctly

Context: DeputyPauseModule.sol#L177-L185

Description: When `pause()` is called, a low-level call to the foundation safe is performed. If the call fails (ie. `success = false`), the error message from the call is cast to a string and bubbled up in a custom error:

```
// Attempt to trigger the call.
(bool success, bytes memory returnData) =
FOUNDATION_SAFE.execTransactionFromModuleReturnData(
    address(deputyGuardianModule), 0, abi.encodeCall(IDeputyGuardianModule.pause,
()), Enum.Operation.Call
);

// If the call fails, revert.
if (!success) {
    revert DeputyPauseModule_ExecutionFailed(string(returnData));
}
```

However, since `returnData` is not a string, casting it to `string` will not bubble up the error correctly and the error message will not be readable.

Recommendation: Consider bubbling up the error message as such:

```
  // If the call fails, revert.
  if (!success) {
-     revert DeputyPauseModule_ExecutionFailed(string(returnData));
+     assembly {
+         revert(add(data, 0x20), mload(data))
+     }
  }
```

Optimism: Acknowledged. This pattern was taken from the `DeputyGuardianModule` and it does work (see `test_pause_targetReverts_reverts` in `DeputyPauseModule.t.sol` for a concrete example of how to assert against the error here).

MiloTruck: Acknowledged.

[L-2] deputyGuardianModule must not be set to the FOUNDATION_SAFE address

Context:

- SelfAuthorized.sol
- DeputyPauseModule.sol#L148-L154
- DeputyPauseModule.sol#L177-L180

Description: In Gnosis Safe, most functions rely on the `requireSelfCall` modifier for caller validation. This means the safe has to perform a self-call in order to call important functions.

In the module, `deputyGuardianModule` can be set to any address by the foundation safe:

```
function setDeputyGuardianModule(IDeputyGuardianModule _deputyGuardianModule)
external {
    if (msg.sender != address(FOUNDATION_SAFE)) {
        revert DeputyPauseModule_NotFromSafe();
    }
    deputyGuardianModule = _deputyGuardianModule;
    emit DeputyGuardianModuleSet(_deputyGuardianModule);
}
```

When `pause()` is triggered, the foundation safe's `execTransactionFromModuleReturnData()` function is called, which will perform a call from the foundation safe to `deputyGuardianModule`:

```
// Attempt to trigger the call.
(bool success, bytes memory returnData) =
FOUNDATION_SAFE.execTransactionFromModuleReturnData(
    address(deputyGuardianModule), 0, abi.encodeCall(IDeputyGuardianModule.pause,
()), Enum.Operation.Call
);
```

However, if `deputyGuardianModule` is wrongly set to the foundation safe, it becomes possible for the deputy to perform a self-call on behalf of the foundation safe.

More specifically, if the foundation safe has a function which:

- Has the `requireSelfCall` modifier
- Has no parameters
- Has a function selector that collides with `pause()`

The deputy would be able to call this function in the foundation safe, which should not be allowed.

Recommendation: As a precautionary measure, consider validating that `deputyGuardianModule` is not the `FOUNDATION_SAFE` address. This check has to be added to both the constructor and `setDeputyGuardianModule()`.

Optimism: Fixed in PR 13594.

MiloTruck: Verified, `deputyGuardianModule` is now checked to not be the `FOUNDATION_SAFE` address before being set.

[L-3] Previous signatures can still be used even when `deputyGuardianModule` is changed

Context: DeputyPauseModule.sol#L163-L167

Description: When `pause()` is called, the caller has to provide a signature from the `deputy` address:

```
// Verify the signature.
bytes32 digest = _hashTypedDataV4(keccak256(abi.encode(PAUSE_MESSAGE_TYPEHASH,
PauseMessage(_nonce))));
if (ECDSA.recover(digest, _signature) != deputy) {
    revert DeputyPauseModule_Unauthorized();
}
```

This allows the deputy to pre-sign transactions, and anyone can call `pause()` on behalf of the deputy should the system need to be paused.

However, as seen from above, the signature digest only includes `_nonce` and does not include the `deputyGuardianModule` address. As such, all signatures from the deputy are valid regardless of what the `deputyGuardianModule` address is set to.

This could be problematic in the following scenario:

- Deputy generates a signature for `deputyGuardianModule = 0x111`.
- Foundation safe sets `deputyGuardianModule` to `0x222`.
- `pause()` can still be called with the deputy's previous signature.

The issue is that authorization from the deputy is not specific to `deputyGuardianModule`. For example, if `deputyGuardianModule` is changed to a malicious contract, the deputy's signature can still be used and there is no way for the deputy to prevent `pause()` from being called.

Recommendation: Consider including `deputyGuardianModule` in the signature digest as well:

```
  struct PauseMessage {
      bytes32 nonce;
+     address deputyGuardianModule;
  }
```

```
- bytes32 internal constant PAUSE_MESSAGE_TYPEHASH = keccak256("PauseMessage(bytes32
nonce)");
+ bytes32 internal constant PAUSE_MESSAGE_TYPEHASH = keccak256("PauseMessage(bytes32
nonce,address deputyGuardianModule)");
```

```
- bytes32 digest = _hashTypedDataV4(keccak256(abi.encode(PAUSE_MESSAGE_TYPEHASH,
PauseMessage(_nonce))));
+ bytes32 digest = _hashTypedDataV4(keccak256(abi.encode(PAUSE_MESSAGE_TYPEHASH,
PauseMessage(_nonce, deputyGuardianModule))));
```

This ensures that all previous signatures from the deputy become invalid once the `deputyGuardian-Module` address changes.

Optimism: Acknowledged. We decide to keep this as-is for simplicity as the risk here is relatively low.

MiloTruck: Acknowledged.

Informational

[I-1] `execTransactionFromModuleReturnData()` does not fail when `deputyGuardianModule` is an address with no code

Context:

- Executor.sol#L22-L24
- DeputyPauseModule.sol#L177-L185

Description: Gnosis safe performs calls using assembly and directly returns the result from `call`. This can be seen in `Executor.execute()`:

```
assembly {
    success := call(txGas, to, value, add(data, 0x20), mload(data), 0, 0)
}
```

In `pause()`, the foundation safe's `execTransactionFromModuleReturnData()` function is called and `success` is checked to be `true` to ensure the call to `deputyGuardianModule` did not fail:

```
// Attempt to trigger the call.
(bool success, bytes memory returnData) =
FOUNDATION_SAFE.execTransactionFromModuleReturnData(
    address(deputyGuardianModule), 0, abi.encodeCall(IDeputyGuardianModule.pause,
()), Enum.Operation.Call
);

// If the call fails, revert.
if (!success) {
    revert DeputyPauseModule_ExecutionFailed(string(returnData));
}
```

However, since `execTransactionFromModuleReturnData()` executes the call in assembly, it will return `success = true` even if `deputyGuardianModule` is an address with no code (eg. an EOA).

Recommendation: Consider adding a comment that `success` will be `true` if `deputyGuardianModule` is an address with no code.

Optimism:: Fixed in PR 13594.

MiloTruck: Verified, the `deputyGuardianModule` address is now checked to contain code before it is set. Additionally, a comment has been added as recommended above.

[I-2] Design choices regarding `pause()`

Context:

1. DeputyPauseModule.sol#L187-L190
2. DeputyPauseModule.sol#L169-L175

Description: This finding is about specific design choices and their impact on the module's functionality.

(1) `pause()` **does not check if the superchain is already paused.** This makes it possible for `pause()` to be called even if the superchain is already paused. Fortunately, there is no impact apart from "burning" the deputy's pre-signed transaction and forcing him to sign a new signature.

(2) **Deputy has no way to invalidate his signatures.** In the module's current implementation, the only way for a nonce to be invalidated is if it was previously used in a signature when `pause()` was called. As such, if the deputy has signed a signature with a certain nonce, there is no way for the deputy to invalidate that signature, which could be needed in some scenarios.

For example:

1. Deputy signs a signature with `nonce = 1`.
2. `pause()` is called with that signature, but it happens to revert for some reason.
3. The superchain no longer needs to be paused and deputy does not want to call `pause()` anymore.
4. However, since the transaction in (2) was broadcasted, anyone can see the deputy's signature and call `pause()` with it.
5. Deputy has no way of invalidating the signature that was broadcasted in (2).

This is not an issue with regular Safe transactions as the nonce can be incremented by signing and executing an empty transaction.

Recommendation: Note that these recommendations do not have to be implemented, they are options that can be considered:

1. Check `SUPERCHAIN_CONFIG.paused() == false` before calling `deputyGuardianModule`.
2. Add a function which allows the deputy to invalidate a nonce (ie. set `usedNonces[_nonce]` to `true`).

Optimism: Acknowledged. For (1), we did this on purpose to avoid adding unnecessary complexity, we'll probably keep it this way for now. For (2), this is actually a defense mechanism where we don't want a compromised key to be able to frontrun signatures and invalidate them.

MiloTruck: Acknowledged.