arbitrary./execution

# LASER WALLET SECURITY ASSESSMENT

**April 21, 2022**

Prepared For:

*Rodrigo Herrera*

Prepared By:

*John Bird*

*Chris Masden*

Changelog:

| | |
|---|---|
| *March 30, 2022* | *Initial report delivered* |
| *April 21, 2022* | *Final report delivered* |

## TABLE OF CONTENTS


TABLE OF CONTENTS ..... 2

EXECUTIVE SUMMARY ..... 4

AUDIT OBJECTIVES ..... 4

OBSERVATIONS ..... 5

SYSTEM OVERVIEW ..... 5

LASERWALLET (SINGLETON) ..... 5

LASERPROXY ..... 6

LASERPROXYFACTORY ..... 6

PRIVILEGED ROLES AND THRESHOLD ..... 6

Regular Owners ..... 6

Special Owners ..... 6

VULNERABILITY STATISTICS ..... 7

FIXES SUMMARY ..... 7

FINDINGS ..... 8

LOW SEVERITY ..... 8

[L1] Unchecked return value ..... 8

[L2] Inconsistent memory cleanup ..... 9

[L3] Incorrect return value on signer retrieval failure ..... 10

[L4] Insufficient checks on LaserWallet implementation contract address ..... 11
[L5] Use of send function is potentially unsafe ..... 12
NOTE SEVERITY ..... 13
[N1] Dead code ..... 13
[N2] Contracts use floating compiler version pragma ..... 13
[N3] Unused domain separator ..... 14
[N4] EIP-4337 not recommended for general use ..... 15
[N5] Unused value in RecoverError enum ..... 15
[N6] Gas savings ..... 16
[N7] Incorrect gas calculations ..... 16
[N8] Inconsistent getter behavior ..... 17
[N9] Unnecessary refund function ..... 17
[N10] Mismatched nonce types ..... 18
[N11] Missing comment in payPrefund() ..... 19
[N12] Typographical Issues ..... 20
APPENDIX ..... 21
APPENDIX A: SEVERITY DEFINITIONS ..... 21
APPENDIX B: FILES IN SCOPE ..... 21

## EXECUTIVE SUMMARY

This report contains the results of Arbitrary Execution's security assessment of Laser Wallet, a multi-signature smart-contract wallet and a fork of Gnosis Safe. It aims to solve pain points that currently exist with using Externally Owned Accounts (EOAs) and multi-signature wallets to control funds. Laser Wallet aims to add extra flexibility to multi-sig wallets by introducing the concept of Special Owners and implementing account abstraction via EIP-4337.

Two Arbitrary Execution (AE) engineers conducted this review over a 3-week period, from March 7, 2022 to March 25, 2022. The audited tag was `1.0.0` (commit hash `d0044ce26c259d6a82bcf050b6c5a2fc25e84f05`) in the `laser-wallet/laser-wallet-contracts` repository. The Solidity files in scope for this audit included all contracts in the `contracts` directory with the exception of contracts in the `test` and `external` directories. The complete list of files is in Appendix B.

A detailed manual review of the codebase was performed with a focus on the new ownership features and account abstraction. In particular, deviations from the original Gnosis Safe implementation received extra scrutiny. In addition to manual review, Slither was used to perform automated static analysis.

The assessment resulted in 17 findings ranging in severity from low to note (informational). Three of the low severity findings were in a modified version of `ecrecover` that contains changes for compatibility with EIP-4337. The other low findings are related to Laser Wallet's migration script, and the way ether is transferred when wallets handle payments. The note findings contain additional observations made during the engagement, as well as suggestions for better adherence to development best practices and gas optimizations.

## AUDIT OBJECTIVES

AE had the following high-level goals for the engagement:

- Ensure that Laser Wallet is implemented consistently with its specification
- Identify smart contract vulnerabilities
- Evaluate adherence to development best practices

The Laser Wallet team also identified specific areas of concern, that focused on deviations from Gnosis Safe behavior. There were three specific areas to investigate: Special Owners, EIP-4337 implementation, and Proxy modifications for EIP-4337 compliance. These areas will be covered in more detail in the System Overview.

AE focused on the introduction of Special Owners, which are a more privileged version of Gnosis Safe owners. A goal of this audit was to identify ways to exploit Laser Wallet's tracking of Special Owners and ensure these new privileges can't be abused by regular owners or non-owners of a wallet.

AE also focused on Laser Wallet's support for EIP-4337. EIP-4337 is a developing standard for account abstraction. There are currently two types of accounts in Ethereum: Externally Owned Accounts (EOAs) and smart contract accounts. The Ethereum protocol requires all transactions (ether transfers or smart contract interactions) to be packaged in a transaction signed by an EOA's private key. Account abstraction aims to reduce

the number of account types down to one by allowing EVM code to not only implement the logic of applications, but also the *verification logic* (managing nonces, signatures, etc.) of individual users' wallets. There are different EIPs with different approaches to implementing this. One proposal, EIP-2938 introduces protocol-level changes to allow transactions to originate from a smart contract account. EIP-4337 achieves account abstraction without making consensus layer protocol changes.

To support EIP-4337, Laser Wallet added two additional functions and made modifications to how it verifies users' signatures. EIP-4337 bans specific opcodes, and this impacted Laser Wallet's proxy implementation. AE was asked to examine their workarounds and identify any potential side effects.

## OBSERVATIONS

Laser Wallet's architecture is similar to Gnosis Safe. The code contains NatSpec documentation and comments throughout the codebase. Many files are nearly identical to Gnosis Safe with the only differences including an updated compiler version pragma, renamed contracts or variables, different error message strings, and additional newlines for better readability. Making changes to vetted code introduces the potential for new bugs, therefore it is recommended to minimize changes when possible.

The Laser Wallet team provided an initial draft of the wallet specification at the start of the engagement. It covers the general architecture and important concepts related to the wallet. The document clearly states that this is a draft and not complete.

It is important to highlight that the EIP-4337 specification is still a work in progress and has not launched on Mainnet. Laser Wallet uses an implementation that is still under development. The standard was recently audited and has changed to address security issues. Laser Wallet will need to account for these changes.

## SYSTEM OVERVIEW

Laser Wallet uses a standard proxy to deploy its smart contract wallets. When a user wants to create a new wallet, they interact with the `LaserProxyFactory` to deploy a new `Proxy` contract. This proxy contract delegates all calls to a base `LaserWallet` implementation contract. The implementation contract address is called the `singleton` and is located at storage slot 0.

### LASERWALLET (SINGLETON)

The `LaserWallet` contract contains the core wallet functionality. It handles first-time setup of the wallet, signature validation for owners, and has the necessary functions to support EIP-4337. It inherits several other contracts described below:

- `EtherPaymentFallback` - Implements a fallback function to receive ether payments
- `Singleton` - Sets up the storage slot for the `singleton` address
- `EntryPoint` - Manages the `EntryPoint` address for EIP-4337
- `OwnerManager` - Tracks and can modify wallet `threshold`, owners, and Special Owners

- `SignatureDecoder` - Handles decoding of signatures into `v`, `r`, and `s` components
- `SecuredTokenTransfer` - Allows the wallet to transfer tokens
- `IERC1271Wallet` - Interface required to be ERC-1271 compliant
- `Enum` - Contains enum definitions
- `Executor` - Responsible for executing wallet transactions
- `IEIP4337` - Interface required to be EIP-4337 compliant
- `ERC1155Holder` - Implements the multi-token standard
- `ERC721Holder` - Interface to support `safeTransferFrom` from ERC721 contracts

## LASERPROXY

A new `LaserProxy` is deployed each time a user creates a wallet. It forwards all transactions to the `singleton` address and relays return data to the user.

## LASERPROXYFACTORY

The `LaserProxyFactory` is responsible for deploying a new proxy contract and executing a message call to the singleton's `setup` function in a single transaction.

## PRIVILEGED ROLES AND THRESHOLD

Laser Wallets have owners, Special Owners and a `threshold`. The `threshold` is the minimum number of authorizations (via `approveHash`) a wallet needs in order to execute a non-Special Owner transaction.

### REGULAR OWNERS

Regular owners, or "owners" for short are standard multi-signature participants. They can authorize wallet transactions with their signature. There must be at least `threshold` authorizations from owners in order for a wallet to execute a non-Special Owner transaction.

### SPECIAL OWNERS

Special Owners are unique to Laser Wallet, and can authorize a transaction with their signature alone. They bypass the `threshold` set in the wallet. Laser wallet was designed in such a way that Special Owners are also considered regular owners by a wallet.

## VULNERABILITY STATISTICS

| Severity | Count |
|---|---|
| Critical | 0 |
| High | 0 |
| Medium | 0 |
| Low | 5 |
| Note | 12 |

## FIXES SUMMARY

| Finding | Severity | Status |
|---|---|---|
| L1 | Low | Fixed in pull request #1 |
| L2 | Low | Fixed in pull request #2 |
| L3 | Low | Fixed in pull request #11 |
| L4 | Low | Fixed in pull requests #13 and #12 |
| L5 | Low | Fixed in pull request #15 |
| N1 | Note | Fixed in pull requests #3 and #4 |
| N2 | Note | Fixed in pull request #16 |
| N3 | Note | Fixed in pull request #14 |
| N4 | Note | Acknowledged |
| N5 | Note | Fixed in pull request #5 |
| N6 | Note | Partially fixed in pull request #6 |
| N7 | Note | Fixed in pull request #15 |
| N8 | Note | Acknowledged |
| N9 | Note | Fixed in pull request #15 |
| N10 | Note | Fixed in pull request #10 |
| N11 | Note | Fixed in pull request #7 |
| N12 | Note | Fixed in pull requests #8 and #9 |

## FINDINGS

### LOW SEVERITY

#### [L1] UNCHECKED RETURN VALUE

The `ecrecover2` function in `ECDSA.sol` is a custom function that was created to perform identically to the `ecrecover` function in the OpenZeppelin `ECDSA.sol` library. For compatibility with EIP-4337, `ecrecover2` does not use the `GAS` opcode.

In this function, a `staticcall` is performed and the result is saved to a `status` variable, but it is never checked to ensure the call was successful:

```
status := staticcall(not(0), 0x01, pointer, 0x80, pointer, 0x20)
```

##### RECOMMENDATION

Consider adding a `require` statement to ensure the call succeeded as expected.

##### UPDATE

Fixed in pull request #1. EIP-4337 has been updated to allow `GAS` opcodes if they are immediately followed by a `*CALL` opcode. As a result, the EIP developers have also removed the custom ECDSA.sol implementation. Consider consulting changes made to the reference implementation as the standard is still under development.

## [L2] INCONSISTENT MEMORY CLEANUP

The `ecrecover2` function in `ECDSA.sol` is a custom function that was created to perform identically to the `ecrecover` function in the OpenZeppelin `ECDSA.sol` library. For compatibility with EIP-4337, `ecrecover2` does not use the `GAS` opcode.

The assembly used in this function cleans up the `hash` and `v` variables stored in memory by setting their location in memory to 0, but it does not cleanup the `r` and `s` variables:

```
mstore(pointer, hash)
mstore(add(pointer, 0x20), v)
mstore(add(pointer, 0x40), r)
mstore(add(pointer, 0x60), s)

...

mstore(pointer, 0)
mstore(add(pointer, 0x20), 0)
```

There is no requirement that the memory must be set to 0 when the location is no longer used. The Solidity documentation states that "The memory may or may not be zeroed out. Because of this, one should not expect the free memory to point to zeroed out memory."

### RECOMMENDATION

Consider removing the cleanup for the `hash` and `v` variables as it is not required and will save on gas usage.

### UPDATE

Fixed in pull request #2.

## [L3] INCORRECT RETURN VALUE ON SIGNER RETRIEVAL FAILURE

The `ecrecover2` function in `ECDSA.sol` is a custom function that was created to perform identically to the `ecrecover` function in the OpenZeppelin `ECDSA.sol` library. For compatibility with EIP-4337, `ecrecover2` does not use the `GAS` opcode.

To derive a `signer` from a signed message, the following are required: - `hash` (signed message) - `v` - `r` - `s`

A `staticcall` is then made to the address `0x1` and if the recovery is successful the signer address will be written to a location specified in the `staticcall` (`pointer` in this case):

```
let pointer := mload(0x40)
mstore(pointer, hash)

...

status := staticcall(not(0), 0x01, pointer, 0x80, pointer, 0x20)
signer := mload(pointer)
```

The problem with the current implementation is that if the `staticcall` to the `0x1` address fails to retrieve the `signer` from a signed message, the last 20 bytes of `hash` will be used as the `signer`. This is inconsistent with the OpenZeppelin library, which returns 0 if it fails to recover the `signer`.

## RECOMMENDATION

In order to replicate the behavior of the `ecrecover` function, consider storing the output of the `staticcall` function in a different memory location than the input, and zeroing out the memory at that location prior to using it.

## UPDATE

Fixed in pull request #11.

## [L4] INSUFFICIENT CHECKS ON LASERWALLET IMPLEMENTATION CONTRACT ADDRESS

`Migration.sol` is a sample contract forked from Gnosis safe. The original file can be found here. It serves as an example for how to change the singleton address of a Gnosis Safe and downgrade it to an earlier version.

The Gnosis Safe changelog contains a warning about this file:

### ADD MIGRATION EXAMPLE TO DOWNGRADE FROM 1.3.0 TO 1.2.0

> File: `contracts/examples/libraries/Migrate_1_3_0_to_1_2_0.sol`
>
> Note: This contract is meant as an example to demonstrate how to facilitate migration in the future. This should not be used in production without further checks.
>
> Expected behaviour:
>
> This migration can be used to migrate a Safe to another singleton address. Once the migration has been executed the singleton address will point to the address specified in the constructor of the migration and the domain separator will be properly set in storage (as this is required by the 1.2.0 version of the Safe contracts).
>
> Note: This is meant as an example contract, only to be used in production if you know what you do.

Outside of a zero address check, this contract makes no other checks to ensure `targetSingleton` points to a valid LaserWallet implementation contract. If a non-LaserWallet contract address is used, funds in the proxy could be put at risk since there is no way to withdraw from the proxy or change its singleton address.

### RECOMMENDATION

- Consider using ERC-165 to provide additional checks on the `targetSingleton` address.
- Consider adding comments to `Migration.sol` to highlight the importance of setting `targetSingleton` to a LaserWallet implementation.

### UPDATE

Fixed. An `isLaser` function was added to `LaserWallet.sol` in pull request #13. With pull request #12, the migration contract now calls `isLaser` before setting the singleton address.

## [L5] USE OF SEND FUNCTION IS POTENTIALLY UNSAFE

The `handlePayment` function in `LaserWallet.sol` uses the following code to send ether to an address:

```
require(receiver.send(payment), "Could not pay gas costs with ether");
```

Using Solidity's built-in `send` function is discouraged as it only forwards a stipend of 2300 gas. Any future gas cost adjustments to opcodes may cause a transfer of ether to exceed 2300 gas.

### RECOMMENDATION

Consider using the built-in `call` function to transfer ether in conjunction with OpenZeppelin's ReentrancyGuard contract which provides the `nonReentrant` modifier that can be applied to either the `execTransaction` function or the `handlePayment` function.

### UPDATE

Fixed. The `handlePayment` function containing `receiver.send(payment)` was removed in pull request #15.

## NOTE SEVERITY

### [N1] DEAD CODE

Changes to signature recovery and checking were necessary to make Laser Wallet compliant with EIP-4337. Functions like `ecrecover` cannot be used because they contain forbidden opcodes. While making changes to `ECDSA.sol` and `LaserWallet.sol`, code was commented out rather than deleted.

- `tryRecover()` has a call to `ecrecover` on line 65 that is commented out.
- `checkNSignatures` has a call to `hash.recover` on line 399 that is commented out.

#### RECOMMENDATION

Consider removing code that is commented out to increase readability.

#### UPDATE

Fixed in pull requests #3 and #4.

### [N2] CONTRACTS USE FLOATING COMPILER VERSION PRAGMA

All Laser Wallet contracts use the following compiler version pragma:

```
pragma solidity ^0.8.9;
```

Locking the compiler version prevents accidentally deploying the contracts with an older Solidity version that may contain unfixed bugs, or with a different version than what was used for testing.

The current pragma prevents contracts from being deployed with an outdated compiler version, but still allows contracts to be deployed with newer non-breaking compiler versions. (up to `0.8.13` at the time of writing). These newer versions may have higher risks of undiscovered bugs.

It is best practice to deploy contracts with the same compiler version that is used during testing and development.

#### RECOMMENDATION

Consider locking the compiler pragma to the specific version used in developing and testing.

#### UPDATE

Fixed in pull request #16.

## [N3] UNUSED DOMAIN SEPARATOR

`Migration.sol` is a sample contract forked from Gnosis Safe. The original file can be found here. It serves as an example for how to change the singleton address of a Gnosis Safe and downgrade it to an earlier version.

Part of the Gnosis downgrade process included changing the `DOMAIN_SEPARATOR_TYPEHASH` as it differs between safe versions. The migration script defines the following typehash:

```
//keccak256(
//    "EIP712Domain(address verifyingContract)"
//);
    bytes32 private constant DOMAIN_SEPARATOR_TYPEHASH =
0x035aff83d86937d35b32e04f0ddc6ff469290eef2f1b692d8a815c89404d4749;
```

This legacy typehash format is still defined in Laser Wallet's `Migration.sol` file. It does not match the typehash in LaserWallet.sol. Furthermore, the variable is unused.

### RECOMMENDATION

Consider removing `DOMAIN_SEPARATOR_TYPEHASH` from `Migration.sol`.

### UPDATE

Fixed in pull request #14.

## [N4] EIP-4337 NOT RECOMMENDED FOR GENERAL USE

EIP-4337 is still under development, and there have been no public audits of existing implementations. The Ethereum Imporovment Proposals page contains a warning:

> This EIP is not recommended for general use or implementation as it is likely to change.

Furthermore, the "Security Considerations" section emphasizes the importance of an audit:

> The entry point contract will need to be very heavily audited and formally verified, because it will serve as a central trust point for all ERC 4337 wallets.

If a vulnerability is discovered in the standard, Laser Wallet may be affected.

### RECOMMENDATION

It is advisable to keep in mind that any changes to EIP-4337 before its Mainnet launch will have to be accounted for in Laser Wallet. Consider limiting the initial deployment of Laser Wallet to a small set of test users until the EIP-4337 standard is finalized.

### UPDATE

Acknowledged, and will not fix. Laser Wallet's statement for this issue:

> We will launch with caution and test it for the time being.

## [N5] UNUSED VALUE IN RECOVERERROR ENUM

In `ECDSA.sol`, the `RecoverError` enum object has an `InvalidSignatureLength` value that is unused. The OpenZeppelin `ECDSA.sol` library uses `InvalidSignatureLength` as an error condition when determining if a signature is packed with the standard methodology (`r`, `s`, `v`) or if it uses the compact signature representation (EIP-2098's secp256k1 signature) (`r`, `yParity`, `s`). This is not needed in the `ECDSA.sol` contract used by `Laser Wallet` because the values of `r`, `s`, and `v` are passed in as function parameters to the `recover` function.

### RECOMMENDATION

Consider removing the `InvalidSignatureLength` value from the `RecoverError` enum.

### UPDATE

Fixed in pull request #5.

## [N6] GAS SAVINGS

In the `setupOwners` function in `OwnerManager.sol` the length of the `_owners` and `_specialOwners` variables are used multiple times, which uses more gas than using a variable on the stack:

- OwnerManager.sol, lines 47-48
- OwnerManager.sol, line 53
- OwnerManager.sol, line 69
- OwnerManager.sol, line 71
- OwnerManager.sol, line 72
- OwnerManager.sol, line 82

### RECOMMENDATION

Consider storing the length of the `_owners` and `_specialOwners` variables each in a distinct variable if it will not impact code readability.

### UPDATE

Partially fixed in pull request #6. In OwnerManager.sol, line 50 `_owners.length` is still used instead of the stored length.

## [N7] INCORRECT GAS CALCULATIONS

The Laser Wallet team requested an investigation of the gas logic in `LaserWallet.sol` to see if it is necessary.

In the `execTransaction` function in `LaserWallet.sol` there is logic that is used to determine whether the amount of gas left in the transaction is sufficient for the remaining tasks that need to execute, and save gas by aborting early if there is not enough gas left. However, this code was forked from a Gnosis Safe contract and contains hard-coded values that are specific to well-bounded operations for that use case. In the Laser Wallet contract the amount of gas required is unknown because it performs a `call` or a `delegatecall` to an arbitrary function in the `execute` function, so this logic cannot determine whether the gas left is sufficient or not.

### RECOMMENDATION

Consider removing the gas calculation logic in the `execTransaction` function and removing the associated `txGas` function parameter from the `execute` function in `Executor.sol`. The existing `txGas` value passed to the `call` and `delegatecall` functions in `execute` can be replaced with `type(uint256).max` to forward all remaining gas to these function calls.

### UPDATE

Fixed in pull request #15. The `Executor.sol` contract was left unmodified, and `gasleft() - 2500` is used as the `txGas` parameter to `execute`. This behavior is consistent with Gnosis Safe.

## [N8] INCONSISTENT GETTER BEHAVIOR

`OwnerManager.sol` contains getter functions for `owners` and `specialOwners` of a Laser Wallet.

The `getOwners` function returns an array of addresses and is unchanged from the Gnosis Safe implementation. The `getSpecialOwners` function behaves differently from `getOwners` in that it will revert if the `specialOwnerCount` value is 0. `getOwners` will always return an array. While there is no impact in this particular case, it is desirable to stay consistent with Gnosis behavior whenever possible.

### RECOMMENDATION

Consider returning an empty array instead of reverting if `specialOwnerCount` is zero.

### UPDATE

Acknowledged by the Laser Wallet team.

## [N9] UNNECESSARY REFUND FUNCTION

**Client reported:** Laser Wallet identified this issue during the audit.

The `handlePayment` function in `LaserWallet.sol` is not required for correct functionality of the wallet. Bundlers submit a batch of user operations associated with different wallets to a miner in a single transaction. Each wallet prefunds its user operation in the bundle via the `payPrefund` function which is called at the end of the `validateUserOp` function. If the prefund amount is too large, any refund will be handled by the wallet's `EntryPoint` contract.

### RECOMMENDATION

Consider removing the `handlePayment` function in `LaserWallet.sol`.

### UPDATE

Fixed in pull request #15.

## [N10] MISMATCHED NONCE TYPES

The `nonce` variable used in `LaserWallet.sol` is of type `uint256`. The `LaserWalletStorage` contract, which replicates the storage of the `LaserWallet` contract, instead uses a `nonce` of type `bytes32`. This mismatch has no security impact, as `bytes32` and `uint256` use the same size storage slot. However, the reference storage contract should be consistent with the types used in the implementation contract.

### RECOMMENDATION

Consider changing the type of `nonce` from `bytes32` to `uint256` in the `LaserWalletStorage` contract.

### UPDATE

Fixed in pull request #10.

## [N11] MISSING COMMENT IN PAYPREFUND()

The `payPrefund` function in `LaserWallet.sol` sends a specified amount of ether to `msg.sender` but does not check to ensure that the call has succeeded:

```
function payPrefund(uint256 _requiredPrefund) internal {
    if (_requiredPrefund > 0) {
        (bool success,) = payable(msg.sender).call{value : _requiredPrefund, gas :
type(uint).max}("");
        (success);
    }
}
```

This looks like a problem, however it is the responsibility of the `EntryPoint` contract to verify that the payment has succeeded. The reference implementation for EIP-4337 includes the following comment that makes this clear:

```
function _payPrefund(uint requiredPrefund) internal {
    if (requiredPrefund != 0) {
        //pay required prefund. make sure NOT to use the "gas" opcode, which is
banned during validateUserOp
        // (and used by default by the "call")
        (bool success,) = payable(msg.sender).call{value : requiredPrefund, gas :
type(uint).max}("");
        (success);
        //ignore failure (its EntryPoint's job to verify, not wallet.)
    }
```

### RECOMMENDATION

Consider adding the `"//ignore failure (its EntryPoint's job to verify, not wallet.)"` comment to alert readers of the `EntryPoint` responsibilities and to make it clear why `success` is not being checked in the wallet.

### UPDATE

Fixed in pull request #7.

## [N12] TYPOGRAPHICAL ISSUES

Three typographical issues were identified during the audit:

- In `Singleton.sol`, line 6: “proxies/GnosisSafeProxy.sol” should be “proxies/LaserWalletProxy.sol”
- In `OwnerManager.sol`, line 154: “cannot be less than” should be “cannot exceed”
- In `TokenTransfer.sol`, lines 40-48: The cases within the switch statement can be indented for better readability

### RECOMMENDATION

Consider addressing the typographical errors found.

### UPDATE

Fixed in pull requests #8 and #9.

## APPENDIX

### APPENDIX A: SEVERITY DEFINITIONS

| Severity | Definition |
|---|---|
| Critical | This issue is straightforward to exploit and is likely to lead to catastrophic impact for client's reputation and can lead to financial loss for client or users. |
| High | This issue is difficult to exploit and is likely to lead to catastrophic impact for client's reputation and can lead to financial loss for client or users. |
| Medium | This issue is important to fix and puts a subset of users' data at risk and is possible to lead to moderate financial impact. |
| Low | This issue is not exploitable in a recurring basis and cannot have a significant impact on execution. |
| Note | This issue does not pose an immediate risk but is relevant to security best practices. |

### APPENDIX B: FILES IN SCOPE

```
./LaserWallet.sol
./base/Executor.sol
./base/OwnerManager.sol
./common/EntryPoint.sol
./common/Enum.sol
./common/EtherPaymentFallback.sol
./common/SecuredTokenTransfer.sol
./common/SelfAuthorized.sol
./common/SignatureDecoder.sol
./common/Singleton.sol
./common/StorageAccessible.sol
./interfaces/IEIP4337.sol
./interfaces/IERC1271Wallet.sol
./libraries/ECDSA.sol
./libraries/LaserWalletStorage.sol
./libraries/UserOperation.sol
./migration/Migration.sol
./proxies/IProxyCreationCallback.sol
./proxies/LaserProxy.sol
./proxies/LaserProxyFactory.sol
```