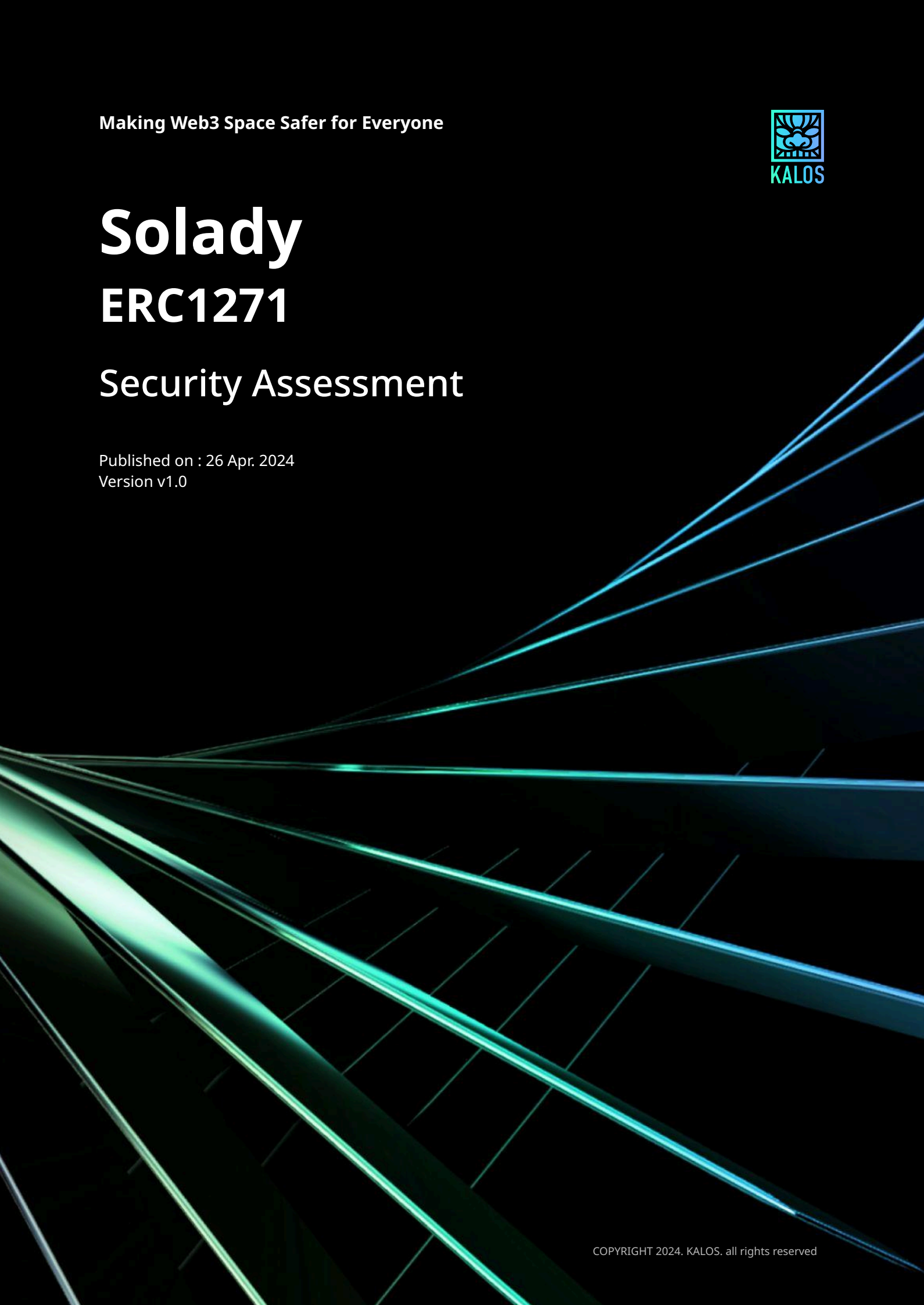
Making Web3 Space Safer for Everyone

KALOS

Solady
ERC1271

Security Assessment

Published on : 26 Apr. 2024
Version v1.0



Security Report Published by KALOS

v1.0 26 Apr. 2024

Auditor: Hun

Found issues

Severity of Issues	Findings	Resolved	Acknowledged	Comment
Critical	-	-	-	-
High	2	1	1	-
Medium	-	-	-	-
Low	-	-	-	-
Tips	-	-	-	-

TABLE OF CONTENTS

ABOUT US

Making Web3 Space Safer for Everyone

KALOS is a flagship service of HAECHI LABS, the leader of the global blockchain industry. We bring together the best Web2 and Web3 experts. Security Researchers with expertise in cryptography, leaders of the global best hacker team, and blockchain/smart contract experts are responsible for securing your Web3 service.

Having secured $60B crypto assets on over 400 main-nets, Defi protocols, NFT services, P2E, and Bridges, KALOS is the only blockchain technology company selected for the Samsung Electronics Startup Incubation Program in recognition of our expertise. We have also received technology grants from the Ethereum Foundation and Ethereum Community Fund.

Inquiries: audit@kalos.xyz
Website: https://kalos.xyz

Executive Summary

Purpose of this report

This report was prepared to audit the security of the Solady ERC1271. KALOS conducted the audit focusing on whether the system created by the ERC1271 is soundly implemented and designed as specified in the materials, in addition to the safety and security of the Solady ERC1271.
In detail, we have focused on the following

- Known Smart Contract vulnerabilities
- Asset theft from the contract
- Signature Verification Bypass
- Memory safety from assembly code
- UI Spoofing on user wallet

Codebase Submitted for the Audit

The codes used in this Audit can be found on GitHub (https://github.com/vectorized/solady/tree/58362f785c90c6693e44f358fac594b0851bb357).

The fixed commit of the code used for this Audit is “https://github.com/vectorized/solady/tree/8e05721ea322821320cc079a88059948c8b3b92d”.

Audit Timeline

Date	Event
2024/4/22	Audit Initiation (ERC1271)
2024/4/26	Delivery of v1.0 report.

Findings

KALOS found 0 Critical, 2 High, 0 Medium, and 0 Low severity issues. There is 0 Tips issue explained that would improve the code's usability or efficiency upon modification.

Severity	Issue	Status
High	Flexible Parent Type causes UI Spoofing on Wallet and leads to Draining User Assets	(Acknowledged - v1.0)
High	Implicit Child Domain Separator in Child Hash leads to Loss of Unexpected User Assets	(Resolved - v1.0)

Remarks

One issue found has not been completely resolved.

OVERVIEW

Protocol overview

• **ERC1271**

This contract provides a signature validation function called *isValidSignature()*. It allows smart contract wallets to verify the signatures of EOA. Solady's ERC1271 employs the nested (wrapped) EIP712. This was recently proposed to protect EIP1271 wallets against replay attack by including the wallet account address in the signed message. For example, in the case of typed data, The final signed message contains two domains (*DOMAIN_SEP_A*, *DOMAIN_SEP_B*) of the account (*Parent*) and application (child), as the parent message wraps the child's typed data hash (*childHash*) and the child's message hash (*child*). The signature validation function reconstructs the child's type data hash with the given child's domain separator and the child's message hash and verifies this to the given child's typed data hash.

```
// For Typed Data
// Parent(bytes32 childHash, Child child)
keccak256(\x19\x01 || DOMAIN_SEP_A ||
    hashStruct(Parent({
        childHash: keccak256(\x19\x01 || DOMAIN_SEP_B || hashStruct(originalStruct)),
        child: hashStruct(originalStruct)
    }))
)

// For Personal Sign
// Parent(bytes32 childHash)
keccak256(\x19\x01 || DOMAIN_SEP_A ||
    hashStruct(Parent({
        childHash: keccak256(\x19Ethereum Signed Message:\n ||
            base10(bytes(someString).length) || someString)
    }))
)
```

[Nested EIP712 Message Format]

Moreover, Solady's ERC1271 provides signature validation through an RPC call. This feature allows wallet clients to use an easy sign method for any message off-chain, like sign-in on the website. This function must be called via RPC and has several guards to prevent abuse calling on-chain,

Scope

```
|       ├── src
|       |   ├── accounts
|       |   |   ├── ERC1271.sol
```

FINDINGS

1. Flexible Parent Type causes UI Spoofing on Wallet and leads to Draining User Assets

ID: SOLADY-01
Severity: High
Type: Visibility
Difficulty: Low

Impact

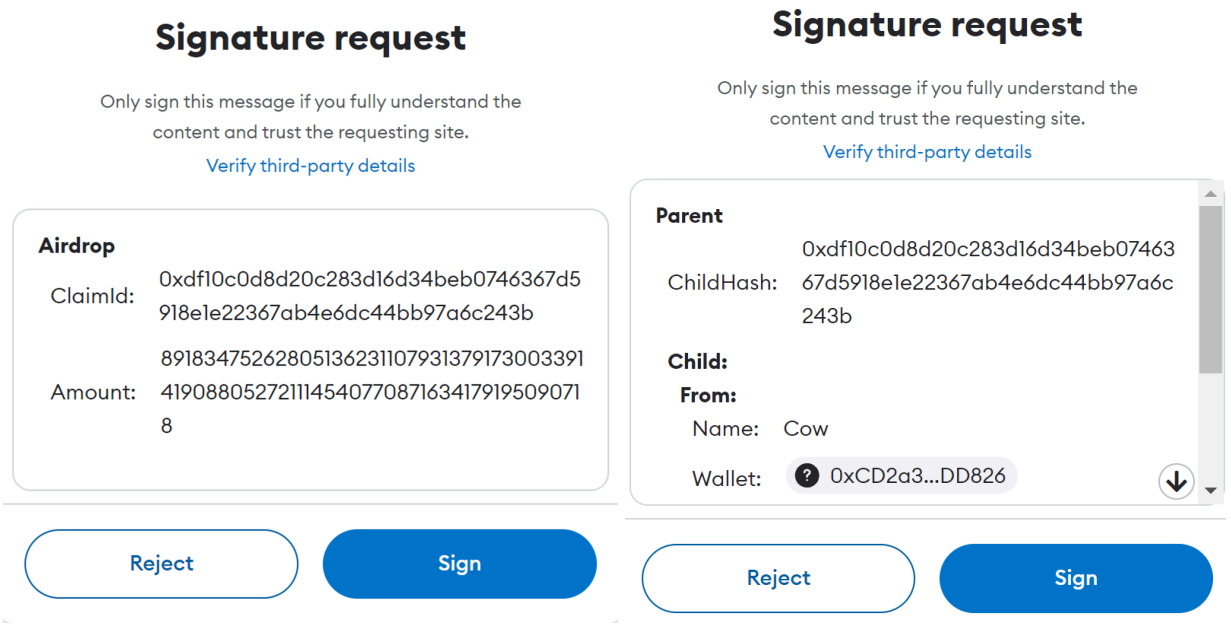
An attacker can lead users to sign hidden malicious message to steal their assets.

Description

ERC1271.isValidSignature() allows any parent type hash since a child struct type had to be flexible to sign various client messages like Permit and Permit2. Nested EIP712 intends *Parent(bytes32 childHash, Child child)Child(...)*, and all types are allowed as the Child type.

If an attacker uses *Parent(bytes32 childHash, bytes32 child)* as a parent type and requests users on compromised or scam websites to sign this parent message where the child is the *Child child*'s hash (Child is the Permit message type), then users will see only hash value instead of struct value and sign it. Finally, their assets will be drained.

Since the parent type is too flexible, attackers can lead a user to sign a parent message for a malicious child message hash.



In the PoC, *Parent* was replaced with *Airdrop*. *bytes32 claimId* is for *bytes32 childHash*, and *uint256 amount* is for *Child child*. All the values are not changed. Even if the type and name differ, the attacker can get a signature for the same child message.

Users should be able to clearly notice the child message they will sign, but they cannot on the maliciously spoofed sign UI, which can lead to the theft of user assets.

Personal sign also suffers from this issue. A user cannot notice the child message to be signed if the parent type is *Parent(bytes32 childHash)*

Recommendation

Force parent type to prevent phishers from hiding malicious child messages exploiting different parent types.

Fix comment

For the personal sign, only *PersonalSign(bytes prefixed)* can be used as the Parent type.
For the typed data, The fix is in progress.

2. Implicit Child Domain Separator in Child Hash leads to Loss of Unexpected User Assets

ID: SOLADY-02	Severity: High
Type: VIsibility	Difficulty: Low

Impact

An attacker can lead users to sign a message with a hash containing a malicious domain (contract) to steal their assets.

Description

For nested EIP712, the parent type for typed data contains only child hash and child. The child domain separator is extracted from a signature parameter. This means attackers can pass arbitrary child domain separator to *isValidSignature()*. Wallet UI will show only the child hash in bytes32 type and the child in struct type without child domain information because the child hash implicitly contains the child domain separator. A domain separator contains the contract address, which can be hidden in childHash.

An attacker can request users on compromised or malicious websites to sign a parent message containing a malicious child domain like the Permit for a high-price token. Users cannot notice what the child domain separator is and will sign it. Eventually, users will lose their unexpected assets.

Recommendation

The child domain separator should be clearly shown on the wallet UI.

Use a parent domain type (hash), which is the same as a child domain type (hash), and include the parent domain in the parent message so that it can be displayed to users.

DISCLAIMER

This report does not guarantee investment advice, the suitability of the business models, and codes that are secure without bugs. This report shall only be used to discuss known technical issues. Other than the issues described in this report, undiscovered issues may exist such as defects on the main network. In order to write secure codes, correction of discovered problems and sufficient testing thereof are required.

Appendix. A

Severity Level

CRITICAL	Must be addressed as a vulnerability that has the potential to seize or freeze substantial sums of money.
HIGH	Has to be fixed since it has the potential to deny users compensation or momentarily freeze assets.
MEDIUM	Vulnerabilities that could halt services, such as DoS and Out-of-Gas, need to be addressed.
LOW	Issues that do not comply with standards or return incorrect values
TIPS	Tips that makes the code more usable or efficient when modified

Difficulty Level

	Low	Medium	High
Privilege	anyone	Miner/Block Proposer	Admin/Owner
Capital needed	Small or none	Gas fee or volatile as price change	More than exploited amount
Probability	100%	Depend on environment	Hard as mining difficulty

Vulnerability Category

Arithmetic	• Integer under/overflow vulnerability • floating point and rounding accuracy
Access & Privilege Control	• Manager functions for emergency handle • Crucial function and data access • Count of calling important task, contract state change, intentional task delay
Denial of Service	• Unexpected revert handling • Gas limit excess due to unpredictable implementation
Miner Manipulation	• Dependency on the block number or timestamp. • Frontrunning
Reentrancy	•Proper use of Check-Effect-Interact pattern. •Prevention of state change after external call • Error handling and logging.
Low-level Call	• Code injection using delegatecall • Inappropriate use of assembly code
Off-standard	• Deviate from standards that can be an obstacle of interoperability.
Input Validation	• Lack of validation on inputs.
Logic Error/Bug	• Unintended execution leads to error.
Documentation	•Coherency between the documented spec and implementation
Visibility	• Variable and function visibility setting
Incorrect Interface	• Contract interface is properly implemented on code.

End of Document