⚠

ERC-2771 Delegatecall Vulnerability

Make sure your contracts are not affected by the recently disclosed vulnerability Please read the following resources from OpenZeppelin and ThirdWeb explaining the vulnerability:

- Arbitrary Address Spoofing Attack: ERC2771Context Multicall Public Disclosure
- Security Vulnerability Incident Report 12/8 *

Vulnerability explained

ERC-2771 is a standard enabling contracts to authenticate users during transaction relaying. Before delving into the security risks of its implementation, it is crucial to understand the mechanics of the ERC-2771 flow.

ERC-2771 Overview

User Request Signing

- The user signs their request and incorporates this signature into the payload.
-

Relay Contract Verification

- The relay contract validates the signature and appends the user's 20-byte address to the end of thecalldata
- .
-

Target Contract Decoding

- The target contract decodes the user address by extracting the last 20 bytes from thecalldata
- , but only whenmsg.sender
- is the relay contract, known as the Trusted Forwarder.
- Decoding is done using assembly for efficiency, as shown in the following code snippet:
-

```

Copy // Decoding the user Address function _msgSender()internalviewvirtualoverridereturns(addresssender) { if(isTrustedForwarder(msg.sender)) { // The assembly code is more direct than the Solidity version using abi.decode. assembly{ sender:=shr(96,calldataload(sub(calldatasize(),20))) } }else{ returnsuper._msgSender(); } }

```

Risks ofdelegatecall

Context Preservation indelegatecall

- When Contract A invokes Contract B usingdelegatecall()
- ,msg.sender
- in Contract B remains the original caller, asdelegatecall()
- preserves the caller's context.
-

Address Extraction in ERC-2771

- As outlined above, extracting the original user address involves verifying thatmsg.sender
- is the Trusted Forwarder, then retrieving the user address from the final 20 bytes ofcallData
- .
-

ERC-2771 Relayer Specifics

- If an ERC-2771 Relayer is employed and the target method usesdelegatecall()
- to its own address (address(this).delegatecall(...)
- ), the Trusted Forwarder check will always pass, asmsg.sender
- will consistently be the Gelato Relay Contract.
- In scenarios where the target method modifies thecalldata
- , it becomes uncertain whether the last 20 bytes accurately represent the original user when_msgSender()
- is invoked. * If you're implementingdelegatecall() in conjunction with ERC-2771, please reach out

to us for assistance. We'll help ensure that your implementation is robust and secure.

Vulnerability conditions

The vulnerability described arises when all three of the following conditions are met in a smart contract. It's crucial to avoid these conditions concurrently.

Avoid the following conditions in the same smart contract:

1. Implementation of ERC2771Context or assumptions on data from the trusted forwarder
2. : the contract either implements ERC2771Context or operates under the assumption that data from the trusted forwarder will be appended to and subsequently extracted from thecalldata
3. .
4. Use of delegatecall to Self-Contract
5. : the contract usesdelegatecall
6. to call itself, typically indicated byaddress(this).delegatecall(...)
7. .
8. Calldata manipulation
9. : situations involving the manipulation ofcalldata
10. , common in functions likemulticall
11. . 12.

Avoid multicall in combination with ERC-2771

The vulnerability is evident in a typicalmulticall function, structured as follows:

```

Copy functionmulticall(bytes[]calldatadata)externalreturns(bytes[]memoryresults) { results=newbytes; for(uinti=0; i<data.length; i++) { (boolsuccess,bytesmemoryresult)=address(this).delegatecall(data[i]); require(success); results[i]=result; } returnresults; }

```

Vulnerability Mechanism

- Within the loop,delegateCall()
- is executed, targeting the contract itself (address(this).delegatecall(data[i]
- ).
- When_msgSender()
- is evaluated within this call, it does not return the original user who signed the transaction. Instead, it yields the last 20 bytes ofdata[i]
- .
-

Potential for Exploitation

- A malicious actor could exploit this by appending a victim's address at the end ofdata[i]
- .
- As a result,_msgSender()
- would erroneously identify the victim's address as the validated user who signed the transaction, leading to potential security breaches.
-

✍ Safemulticall & ERC-2771 implementation

To securely implementmulticall in conjunction with ERC-2771, it is recommended to manually append the context to eachdata[i] , as outlined inOpenZeppelin's blog . The approach involves the following steps:

```

Copy functionmulticall(bytes[]calldatadata)externalreturns(bytes[]memoryresults) { bytesmemorycontext=msg.sender==_msgSender() ?newbytes(0) :msg.data[msg.data.length-20:]; results=newbytes; for(uinti=0; i<data.length; i++) { (boolsuccess,bytesmemoryresult)= address(this).delegatecall(bytes.concat(data[i],context)); require(success); results[i]=result; } returnresults; }

```

Key Points

- Context Determination

- : The context is derived by comparingmsg.sender
- and_msgSender()
- . If they match, no additional context is appended. Otherwise, the last 20 bytes ofmsg.data
- are used.
- Secure Delegatecall
- : By appending the context to eachdata[i]
- before thedelegatecall
- , the function ensures that the original sender's address is correctly interpreted in subsequent calls.
- Robust Error Handling
- : The use ofrequire(success)
- after eachdelegatecall
- ensures that any call that fails will halt the execution, maintaining the integrity of the operation.
-

Previous Security Considerations Next Installation Last updated3 months ago On this page *Vulnerability explained * Vulnerability conditions * _____ Avoid multicall in combination with ERC-2771 * ✅ Safe multicall & ERC-2771 implementation