ERC-2771 (recommended)

Native meta transactions with top notch security If you plan to use ERC-2771 with amulticall method or any other method usingdelegateCall() Please read carefully the sectionAvoid ERC-2771-risks If you are using@gelatonetwork/relay-sdk v3 or contracts from the package@gelatonetwork/relay-context v2 please follow thismigration guide to migrate to the new versions. After reading this page:

- You'll understand the difference betweensponsoredCallERC2771
- and
- callWithSyncFeeERC2771
- You'll understand how to usesponsoredCallERC2771
- andcallWithSyncFeeERC2771
- in combination withERC2771Context
- to achieve a gasless UX for your app, with secure user signature verification.
- You'll understand ERC-2771'score functionality
- and how it allows for the off-chain sender address to be verified on-chain. *

Recommendation for using ERC-2771

As detailed in theSecurity Considerations section, it's crucial to ensure that your relay implementation is impervious to vulnerabilities when using a relayer. The most secure approach is to utilize our ERC-2771 implementations:

- sponsoredCallERC2771
- callWithSyncFeeERC2771
-

When usingsponsoredCallERC2771 , you sponsor your user's gas fees, leveraging1Balance for payment. In contrast, withcallWithSyncFeeERC2771 , the fees are paid from the target contract.

In both instances, users are prompted to sign their transaction's relay request using their private keys (for instance, through MetaMask). This step is crucial for security purposes. Gelato verifies on-chain that the user's signature corresponds with the required address before forwarding the call.

When relaying a message to a target smart contract function, it's essential for the function to authenticate the message's origin and confirm it was forwarded through the correct relayer. Without these verifications, your target function becomes susceptible to exploitation.ERC-2771 employs sophisticated data encoding to relay the original_msgSender from off-chain, and it guarantees that only thetrustedForwarder is capable of encoding this value. These two parameters, in tandem, safeguard against any potential misconduct, ensuring a secure transmission of information from off-chain to on-chain!

Why is this important?

In the context of relaying,msg.sender loses its usual informational significance. Under normal circumstances,msg.sender would denote the user initiating the transaction; however, with off-chain relaying, we lose this valuable piece of information.

Consider this scenario: how does a target smart contract determine who can call a particular function? In this case,msg.sender will be the relayer, but merely whitelisting this address is insufficient and still permits others using the same relayer to call your function. This situation can raise significant concerns, particularly when low-level calls are involved.

The optimal solution would be to allow the initiator of the relay call to specify an address and relay this address on-chain. The target smart contract can then authenticate a function call using this address.

The challenge then becomes: how can we successfully transmit information (a specific address) via low-levelcalldata from off-chain to on-chain without disrupting thecalldata 's integrity?

Core Functionality of ERC-2771

Here's where the real magic unfolds. ThetrustedForwarder encodes thefrom address (i.e., the off-chain address) into thecalldata by appending it at the end:

```

Copy (boolsuccess,)=to.call.value(value)(abi.encodePacked(data,from));

```

Now, the target contract can validate thefrom address by decoding the data in the same manner, ensuring that this message has been passed through thetrustedForwarder .

The necessary target contract function can then confidently confirm that the correct entity signed and requested this payload to be relayed, and only via a trusted forwarder - in our case, the Gelato Relay.

How does Gelato encode this data?

Let's take as an example relay methodsponsoredCallERC2771 . MethodcallWithSyncFeeERC2771 works similarly.

Gelato Relay'ssponsoredCallERC2771 function encodes the user's address, which can then be utilized by the ERC-2771 compatibletarget smart contract. The most relevant part, where the user address is appended to thecalldata , is shown below:

```

GelatoRelay1BalanceERC2771.sol Copy _call.target.revertingContractCall( _encodeERC2771Context(_call.data,_call.user), "GelatoRelay1BalanceERC2771.sponsoredCallERC2771:" );

```

where_encodeERC2771Context refers to:

```

GelatoRelayUtils.sol Copy function_encodeERC2771Context(bytescalldata_data,address_msgSender) pure returns(bytesmemory) { returnabi.encodePacked(_data,_msgSender); }

```

We are encoding thecalldata and the user address together by simply appending the user's address to the end as required by ERC-2771.

How can I modify my smart contract to be ERC-2771 compatible?

Let's take a look at an example using relay methodsponsoredCallERC2771 . ForcallWithSyncFeeERC2771 please refer to the steps describedhere .

1. Install Gelato'srelay-context package in your contract repo

See also relay-context-contracts :Installation

```

Copy npm install --save-dev @gelatonetwork/relay-context

```

or

```

Copy yarn add -D @gelatonetwork/relay-context

```

1. Import theERC2771Context contract:

```

Copy import{ ERC2771Context }from"@gelatonetwork/relay-context/contracts/vendor/ERC2771Context.sol";

```

This contract's main functionality (originally implemented byOpenZeppelin) is to decode the off-chainmsg.sender from the encoded calldata using_msgSender() .

ERC2771Context.sol

```

Copy // SPDX-License-Identifier: MIT // OpenZeppelin Contracts (last updated v4.7.0) (metatx/ERC2771Context.sol)

pragmasolidity^0.8.9;

import"../utils/Context.sol";
```

/ *@devContext variant with ERC2771 support.* / abstractcontractERC2771ContextisContext{ addressprivateimmutable_trustedForwarder;

constructor(addresstrustedForwarder) { _trustedForwarder=trustedForwarder; }

functionisTrustedForwarder(addressforwarder)publicviewvirtualreturns(bool) { returnforwarder==_trustedForwarder; }

function_msgSender()internalviewvirtualoverridereturns(addresssender) { if(isTrustedForwarder(msg.sender)) { // The assembly code is more direct than the Solidity version using abi.decode. /// @solidity memory-safe-assembly assembly{ sender:=shr(96,calldataload(sub(calldatasize(),20))) } }else{ returnsuper._msgSender(); } }

function_msgData()internalviewvirtualoverridereturns(bytescalldata) { if(isTrustedForwarder(msg.sender)) { returnmsg.data[:msg.data.length-20]; }else{ returnsuper._msgData(); } } }

```

- ThetrustedForwarder
- variable is set in the constructor which allows for setting a trusted party that will relay your message to yourtarget
- smart contract. In our case, this isGelato Relay1BalanceERC2771.sol
- which you can find in thecontract addresses
- section.
- The_msgSender()
- function encapsulates the main functionality of ERC-2771, by decoding the user address from the last 20 bytes of thecalldata
- .
-
    - In Solidity, the logic is equivalent to:
- *
-

```

Copy abi.decode(msg.data[msg.data.length-20:], (address));

```

- Gelato's smart contracts handle the encoding of important information to thecalldata
- (seeHow does Gelato encode this data?
- ). It is the job of yourtarget
- smart contract function to decode this information using this_msgSender()
- function.
- The function_msgData()
- removes themsg.sender
- from the entirecalldata
- if the contract was called by thetrustedForwarder
- , or otherwise falls back to return the originalcalldata
- .
-

1. Replace msg.sender with _msgSender()

Within the function that you would like to be called with Gelato Relay, replace all instances ofmsg.sender with a call to the_msgSender() function inherited fromERC2771Context ._msgSender() is the off-chain signer of the relay request, allowing for secure whitelisting on your target function.

1. (Re)deploy your contract and whitelist GelatoRelay1BalanceERC2771

If your contract is not upgradeable, then you will have to redeploy your contract to setGelatoRelay1BalanceERC2771.sol as yourtrustedForwarder :

GelatoRelay1BalanceERC2771.sol is immutable for security reasons. This means that once you setGelatoRelay1BalanceERC2771.sol as your trusted forwarder, there is no way for Gelato to change the ERC2771 signature verification scheme and so you can be sure that the intended_msgSender is correct and accessible from within your target contract.

Please refer to thecontract addresses section to find out which Gelato relay address to use as atrustedForwarder . UseGelatoRelay1BalanceERC2771.sol address forsponsoredCallERC2771 .

Previous Quick Start Next sponsoredCallERC2771 Last updated2 months ago On this page *Recommendation for using ERC-2771 * Why is this important? * Core Functionality of ERC-2771 * How does Gelato encode this data? * How can I modify my smart contract to be ERC-2771 compatible? * 1. Install Gelato's relay-context package in your contract repo* 2.
```

Import the ERC2771Context contract: * 3. Replace msg.sender with _msgSender() * 4. (Re)deploy your contract and whitelist GelatoRelay1BalanceERC2771