transaction-data

A guide to reading transaction input data

In this article, we'll learn how to read and decode the raw calldata that's passed to Ethereum smart contracts. The calldata contains the function signature and arguments passed to the function.

We can start with something simple like0xb7760c8f0000000000000000000000000000000000000000000000000000000000000004d866d9200000000000000000000000068b3465833fb72a70ecdf485e0e4c7bd8665fc45

To start, we need to extract the function signature from the first four bytes of the calldata. These four bytes represent the function selector, which uniquely identifies each function in the contract. We can use a web3 library to extract the function selector and match it to the corresponding function in the contract. If you don't have this, you can just use a tool likehttps://4byte.directory/ to look up the function selector.

Once we have the function selector, we can use the function signature to determine the number and type of arguments that are passed to the function. For example, the function signature for the transfer function in the ERC20 standard is 0xa9059cbb, which corresponds to the function transfer(address to, uint256 value). The first argument is an address, which is 20 bytes long, and the second argument is a uint256, which is 32 bytes long. We can use this information to extract the arguments from the calldata.

// transfer(uint256 amount, address to) 0xb7760c8f // arguments 0000000000000000000000000000000000000000000000000000000004d866d9200000000000000000000000068b3465833fb72a70ecdf485e0e4c7bd8665fc45 Now that the function selector and arguments are separated, the arguments can be decoded. In their most basic form, every argument is padded to 32 bytes. This is because the calldata is stored in 32 byte slots. We can use this information to extract the arguments from the calldata.

// uint256 amount 0000000000000000000000000000000000000000000000000000000004d866d92 // address to 0000000000000000000000068b3465833fb72a70ecdf485e0e4c7bd8665fc45 This is a lot more readable now. The amount is0x4d866d92 and the address is0x68b3465833fb72a70ecdf485e0e4c7bd8665fc45

Dynamic variables

Transaction data containing information that is not a fixed size is much harder to decode. These types would be something like a string or bytes.

To show how complicated strings can be, consider the following example of "hello world" encoded as a parameter.

0x 0000000000000000000000000000000000000000000000000000000000000020 000000000000000000000000000000000000000000000000000000000000000c 48656c6c6f20576f726c6421000000000000000000000000000000000000000000000000 The first 32 bytes represent the offset to the string. The second 32 bytes represent the length of the string. The remaining bytes represent the string itself. The string is padded to 32 bytes, so the remaining bytes are 0s.

This is a basic outline of decoding transaction data by hand. There are many more edge cases and complications that can arise, but this should give you a good idea of how to decode transaction data. If you want to learn more, I recommend reading thishttps://degatchi.com/articles/reading-raw-evm-calldata/ article. This was heavily referenced and has a lot more information to it. Edit this page Previous What is MEV?