**Decoding Ethereum transactions**

**Intro to Function Selectors**

Continuing from the last lesson, when we call the fund function our MetaMask is going to pop up with a bunch of information about the transaction.

By clicking the Hex tab, we can confirm the raw data for this transaction and exactly which function is being called.

We'll go into function selectors a lot more later, but the important thing to understand is that when a Solidity contract is compiled, our functions are converted into a low-level bytecode called a function selector.

When we call our fund function, this is converted to a function selector that we can actually verify using Foundry's cast command.

cast sig "fund()"

The above should result in the output 0xb60d4288 and when we compare this to the Hex data in our MetaMask, we see that it does indeed match!

Were the function being called something secret/nefarious like stealMoney(). This function selector would be completely different. Running our cast command again confirms this clearly with a return of 0xa7ea5e4e.

We can use this knowledge to verify any function we're calling through our browser wallet by comparing the expected and actual function selectors for the transaction.

There's even a way to decode calldata using the cast command.

Let's say our function was a little different and it required an argument.

function fund(uint256 amount) public payable {

require(amount.getConversionRate(s\_priceFeed) >= MINIMUM\_USD, "You need to spend more ETH!");

// require(PriceConverter.getConversionRate(msg.value) >= MINIMUM\_USD, "You need to spend more ETH!");

s\_addressToAmountFunded[msg.sender] += amount;

s\_funders.push(msg.sender);

}

If we were to call this function, the information MetaMask gives us is a little different.

In this instance, we can use the command cast --calldata-decode <SIG> <CALLDATA> to provide us the parameters being passed in this function call!

cast --calldata-decode "fund(uint256)" 0xca1d209d000000000000000000000000000000000000000000000000016345785d8a0000

The above decodes to:

100000000000000000 [1e17]

0.1 Eth! The same amount being passed as an argument to our fund call. It seems this function is safe!

**Wrap Up**

This more or less summarizes how transactions work through our browser wallet and what we can expect to see from a low-level with respect to the encoded function selectors and calldata, we'll go over those in more detail later.

I encourage you to experiment with the remaining functions on the front end. A few things to try:

* Funding and Withdrawing with an account
* Funding with Account A and Withdrawing with Account B - what happens?
* Verify the function selectors of our other functions

In our next lesson we'll recap everything we've learnt so far 💪