Blockchain explorer usage

A blockchain explorer is a crucial tool for viewing, analyzing, and understanding data on a blockchain. It transforms complex, encoded information into human-readable form, making it easier to track transactions, verify addresses, explore smart contracts, and more. Without a blockchain explorer, transaction data and smart contract interactions appear as hex-encoded information, which is not human-readable. For example, calling a Uniswap contract to swap tokens would look like a series of encoded hexadecimal strings. An explorer decodes this data, showing details like token names, transfer amounts, and method names, so users can understand what actions are taking place.

Decoding is done programmatically using a set of rules known as the Ethereum Application Binary Interface (ABI). The ABI defines how to encode and decode function names and parameters for Ethereum smart contracts, allowing a blockchain explorer or other tools to convert raw transaction data into human-readable formats. Several libraries make ABI decoding relatively straightforward, especially for developers building tools or explorers: web3.js, ethers.js, Web3.py.

For common contracts like Uniswap or ERC-20 tokens, the ABI files are usually public and can be added to the explorer's ABI database. These ABIs can be used to automatically identify functions and parameters, enabling accurate decoding.Blockchain explorers identify known smart contracts, such as Uniswap, by using their contract address and displaying relevant details, like contract name and functions. This allows users to recognize trusted contracts versus unknown ones, adding a layer of security when interacting with decentralized finance (DeFi) apps and other dApps.

Explorers like Etherscan and Otterscan can show verified source code and contract metadata (if available) to give insights into the contract’s purpose and functionality. In analysing a transaction sent to a Uniswap contract to perform a token swap the destination address (Uniswap’s contract) and the hex-encoded transaction data (e.g., `0x18cbafe5` followed by more data) couldn’t be interpreted easily. An explorer decodes this, revealing that `0x18cbafe5` maps to `swapExactTokensForTokens` and decodes other parameters, showing token addresses, amounts, and recipient addresses.

In appendix xxx we show how to decode transactions in a purely programmatically way. This proofs the utility of explorers and shows what we need to do when explorer are not sufficient.

Explorer selection

Apart from the de facto standard, which is etherscan, a proprietary close code solution, there are several open-source and free applications that can be installed locally on a self-run Ethereum node. These applications offer functionality similar to Etherscan, the most famous publicly available with limited functionality, including the ability to explore blockchain data, track transactions, and interact with smart contracts. We need to find one implementation well coupled with an Erigon node and with minimal disk requirements and for this we have analysed the most known free open source solutions. BlockScout is a versatile and open-source blockchain explorer. Which supports Ethereum and other Ethereum-compatible networks and can be installed on a local Ethereum node to provide Etherscan-like functionality. Etherchain Explorer is another open-source project that offers blockchain exploration functionality. It's designed to work with Ethereum nodes and provides a web interface to explore blockchain data. Ethereum-ETL is a tool for extracting, transforming, and loading Ethereum blockchain data. While it is more focused on data extraction and transformation, it can be used in conjunction with other tools to build a custom blockchain explorer. Otterscan is a lightweight, open-source Ethereum block explorer embedded in erigon and available as standalone application. All the mentioned explorers, except otterscan require to install an additional DB and rely on ETL to transform chain data and load them in their schema. Otterscan relies on erigon internal mdbx db and, unless instructed differently, uses the erigon indexing process, therefore adding ZERO disk space to the synchronised node. It is especially appealing for developers and researchers who want to avoid reliance on third-party services. Unlike some explorers that require high bandwidth or storage, Otterscan’s design optimizes for performance and fast data retrieval.

Analyse tx in blockchain explorers

When analyzing complex transactions like flashloans in Otterscan and every other explorer, several challenges arise related to the chronological order and detailed information about the loan and its repayment. These challenges stem from how Ethereum transactions and internal complex transactions, such as flashloans, often involve multiple internal transactions and contract calls that happen within a single Ethereum transaction. These internal transactions are not directly recorded on the blockchain like regular external transactions but are instead part of the execution trace of a smart contract.

Flash loans typically involve borrowing, using the borrowed funds for certain operations (e.g., arbitrage or liquidation), and then repaying the loan—all within one transaction.

Otterscan tries to display these internal transactions, but the order may appear out of sync due to how the logs and traces are processed.

Logs vs. Traces Ethereum transactions generate logs (also known as events) and execution traces. Logs are emitted by smart contracts to signal important events (such as a loan being issued or repaid), while traces show the step-by-step execution of the transaction, including internal calls between contracts.

Logs may not always provide enough detail to fully understand the sequence of events, especially in complex transactions. Traces provide more granular detail, but they can be harder to parse, as they include every step of contract execution, which can be nested and convoluted.

Flash loans are atomic by nature—they either fully succeed (with loan repayment) or revert (fail entirely). This atomicity can make it difficult to see the chronological steps in a high-level interface like Etherscan or Otterscan because everything happens within a single block