# OpenTracer: A Dynamic Transaction Trace Analyzer for Smart Contract Invariant Generation and Beyond

Zhiyang Chen
University of Toronto
Toronto, Canada
zhiychen@cs.toronto.edu

Ye Liu
Singapore Management University
Singapore, Singapore
yeliu@smu.edu.sg

Sidi Mohamed Beillahi
University of Toronto
Toronto, Canada
sm.beillahi@utoronto.ca

Yi Li
Nanyang Technological University
Singapore, Singapore
yi_li@ntu.edu.sg

Fan Long
University of Toronto
Toronto, Canada
fanl@cs.toronto.edu

## ABSTRACT

Smart contracts, self-executing programs on the blockchain, facilitate reliable value exchanges without centralized oversight. Despite the recent focus on dynamic analysis of their transaction histories in both industry and academia, no open-source tool currently offers comprehensive tracking of complete transaction information to extract user-desired data such as invariant-related data. This paper introduces OpenTracer, designed to address this gap. OpenTracer guarantees comprehensive tracking of every execution step, providing complete transaction information. OpenTracer has been employed to analyze 350,800 Ethereum transactions, successfully inferring 23 different types of invariant from predefined templates. The tool is fully open-sourced, serving as a valuable resource for developers and researchers aiming to extract or validate new invariants from transaction traces. A demonstration video of OpenTracer is available at https://youtu.be/vTdmjWdYd30. The source code of OpenTracer is available at https://github.com/jeffchen006/OpenTracer.

## CCS CONCEPTS

• **Security and privacy → Software security engineering**; • **Software and its engineering → Software testing and debugging**.

## KEYWORDS

invariant generation, dynamic analysis, smart contract

**ACM Reference Format:**
Zhiyang Chen, Ye Liu, Sidi Mohamed Beillahi, Yi Li, and Fan Long. 2024. OpenTracer: A Dynamic Transaction Trace Analyzer for Smart Contract Invariant Generation and Beyond. In *39th IEEE/ACM International Conference on Automated Software Engineering (ASE '24), October 27-November 1, 2024, Sacramento, CA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3691620.3695361


*ASE '24, October 27-November 1, 2024, Sacramento, CA, USA*

ACM ISBN 979-8-4007-1248-7/24/10
https://doi.org/10.1145/3691620.3695361

## 1 INTRODUCTION

Blockchain technology has transformed global decentralization, with smart contracts being one of its key applications. The transaction histories of these smart contracts capture all execution data since their deployment, providing a rich source of information for analysis, such as dynamic invariants and user behaviors.

OpenTracer is a dynamic analysis tool designed to parse raw transaction traces into human-readable formats, enabling deep analysis of specific trace segments to extract critical data, such as those related to invariants. This tool has been used in the recently accepted work, Trace2Inv [7], at The ACM International Conference on the Foundations of Software Engineering (FSE) 2024. In Trace2Inv, OpenTracer was utilized to analyze 350, 800 Ethereum transactions, from which it extracted pertinent data for 23 different types of invariant templates across 8 categories. It operates by parsing transaction trace data from an Ethereum archive node, subsequently locating and applying dynamic taint analysis and data flow analysis on specific trace snippets to extract the information desired by users. Throughout the development of OpenTracer, we have observed its capability to extract any detail from execution traces, establishing it as an invaluable tool for researchers aiming to generate, validate, and explore new invariants in future studies.

Existing tools related to OpenTracer are typically either closed-source or are tailored to meet specific, predefined requirements(see Section 2). In contrast, OpenTracer is fully open-source and offers extensive customization options, enabling researchers to extract any information from transaction traces. This flexibility makes OpenTracer an exceptionally versatile tool suited to a wide range of research needs. In this paper, we highlight three major applications of OpenTracer:

- As a stand-alone transaction explorer to generate function-level invocation trees of transactions with decoded storage accesses.
- As a dynamic analysis tool to perform deeper analysis (taint analysis/data flow analysis) on specific trace snippets to collect invariant-related data.
- As a replacement for modified archive Ethereum nodes in other tools.

## 2 RELATED WORKS

In this section, we review the existing tools on transaction analysis, identifying the gaps that OpenTracer addresses.

## 2.1 Transaction Explorers

Numerous industry transaction explorers are currently available, offering free services that allow users to delve into transaction details on various blockchains, as detailed in Table 1. While these tools effectively provide basic transaction data at scale, they often fall short in offering comprehensive low-level transaction details, which are critical for program analysis. For instance, except for EthTx [16], all transaction explorers are closed-source. EthTx, though open-source, does not provide storage access information, thus limiting its functionality significantly. In contrast, OpenTracer is fully open-source and equipped to furnish a complete spectrum of transaction data, ensuring that users can extract and utilize the exact information they need from transactions.

**Table 1: Comparison of Transaction Explorers: "Source" represents open-source availability. "Func." represents the display of function level invocation trees. "Store." represents storage access information visibility, including sload and sstore.**

| Explorer | Open Source? | Func. | Store. |
|---|---|---|---|
| Phalcon [5] | ✗ | ✓ | ✗ |
| EthTx [16, 17] | ✓ | ✓ | ✗ |
| AnyTx [15] | ✗ | ✓ | ✓ |
| Tenderly [14] | ✗ | ✓ | ✓ |
| OpenChain [11] | ✗ | ✓ | ✓ |
| OpenTracer | ✓ | ✓ | ✓ |

## 2.2 Dynamic Invariant Generation

Several notable works in the field focus on the generation of invariants from smart contract transaction histories. SPCon [9] reconstructs likely access control models by analyzing function callers in historical transactions, while InvCon [8] and its followup work, InvCon+ [10], employ pre/post conditions to derive invariants specifically designed to counteract prevalent vulnerabilities in smart contracts. However, these methods typically extract only contract state variables and function calls from the transaction logs. By employing OpenTracer, our approach can enhance the implementation of these existing methodologies by extracting more comprehensive data(see Section 3) from the transactions, thus potentially increasing the robustness and applicability of the generated invariants. Additionally, some other research focuses on inferring invariants directly from the contract source code. However, these approaches are unable to capture user behaviors and other dynamic information extracted from transaction traces. In contrast, OpenTracer can capture such dynamic information, which is crucial for generating more accurate and effective invariants.

## 2.3 Transaction Anomaly Detection

Significant efforts have been devoted to detecting anomalous transactions in blockchain environments, with key contributions such as TxSpector [21], Time-Travel Investigation [20], Sereum [13], SODA [6], and The Eye of Horus [18]. All of these approaches, except for The Eye of Horus, require modifications to the Geth client to access necessary execution traces, leading users to adopt a non-standard version of Geth, which must be updated with each new Geth release. Conversely, the Geth client offers a built-in debug RPC method through the `debug_traceTransaction` RPC call, which allows for the replaying of transactions to retrieve execution traces. This feature is embedded in all Geth versions and stands as a vital RPC endpoint. Utilizing this RPC method can avoid the need for a modified Geth client. However, developers are required to parse the returned results, often requiring more development effort than modifying the Geth node itself. Notably, The Eye of Horus also utilizes the `debug_traceTransaction` RPC call but lacks in providing a versatile tool for general transaction analysis. In contrast, OpenTracer not only harnesses this functionality to offer a general-purpose transaction analysis tool but also can possibly replace modified archive nodes for other applications, as demonstrated in Section 4, where OpenTracer serves as an alternative to archive nodes for TxSpector [21].

# 3 OPENTRACER OVERVIEW

OpenTracer is a robust tool designed to meticulously capture and analyze transaction traces. As shown in Figure 1, the process begins when OpenTracer receives a transaction hash. The following steps are executed: (1) OpenTracer downloads the raw trace data from an Ethereum Archive Node. (2) parses this data to construct an invocation tree, (3) decodes function names, arguments, and return data utilizing function ABIs, and decodes storage accesses through dynamic storage tracking, (4) finally analyzes specific trace snippets to extract user-desired data such as invariant-related data.

## 3.1 Download and Augment Transaction Traces

OpenTracer utilizes the Ethereum archive node RPC method `debug_traceTransaction` to download transaction traces. These traces, referred to as *structLogs* and highlighted in Box 1 of Figure 1, consist of a series of executed EVM instructions. Each instruction is detailed in tuple format, encapsulating the program counter, opcode, gas metrics, and the current states of the EVM stack and possibly memory. Additionally, OpenTracer retrieves the transaction receipt using the `eth_getTransactionReceipt` method, which supplements the trace with vital transaction details such as the block number and origin address. These RPC methods are supported by several providers [4, 12], allowing users to use OpenTracer without maintaining a local Ethereum archive node, which is resource-intensive and requires substantial storage space.

## 3.2 EVM Trace Parser

The parser module parses Ethereum Virtual Machine (EVM) trace data to construct an invocation tree. The parser identifies function entry and exit points using specific EVM opcodes categorized as "Function Enter" and "Function Exit". The "Function Enter" opcodes include `call`, `callcode`, `staticcall`, `delegatecall`, `create`, and `create2`, marking the start of a function invocation. Conversely, "Function Exit" opcodes such as `stop`, `return`, `revert`, `selfdestruct`, an invalid opcode and scenarios where execution halts due to running "out of gas" signify the end of a function call. This classification helps in constructing an invocation tree where each node is a detailed tuple recording the contract address, function selector, raw call data and return data. The parser also logs raw `sload` and `sstore` operations, setting the stage for more sophisticated decoding in subsequent steps. An example of the output from this process is depicted in Box 2 of Figure 1.

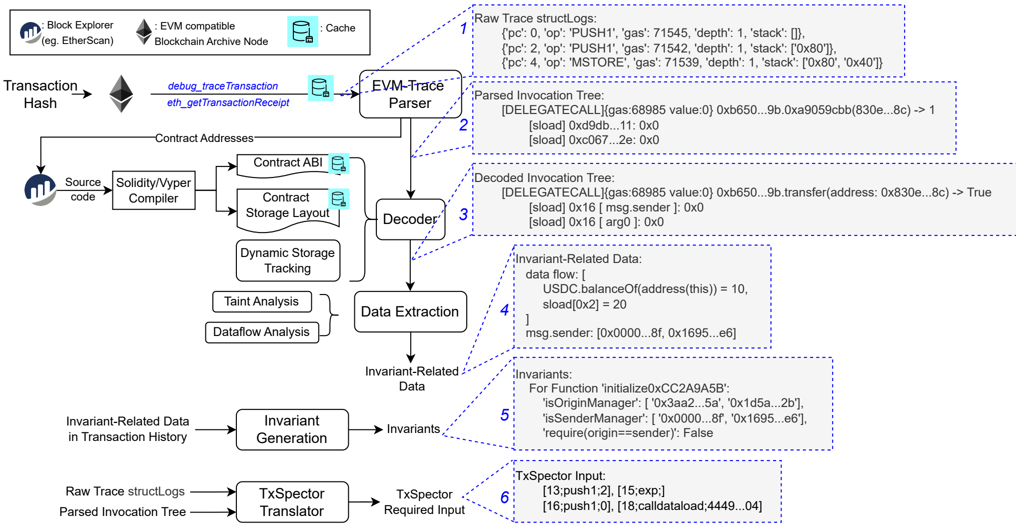


**Figure 1: An Overview of OpenTracer**

## 3.3 Decoder

The decoder module enhances the interpretability of transaction trace data by processing function call data and storage accesses within the invocation tree. A sample output from this module is illustrated in Box 3 of Figure 1, where both raw call data, raw return values, and storage accesses are decoded to more accessible formats.

**Decoding Function Call Data and Return Data.** Each node within the invocation tree is processed by the decoder, which accesses the contract's source code and ABI from blockchain explorers such as EtherScan [3]. The source code is compiled using Solidity [1] or Vyper [2] compilers to verify function signatures. This allows the decoder to translate the raw call and return data into a format that is not only human-readable but also accurately reflects the verified function signatures.

**Decoding Storage Accesses with Dynamic Storage Tracking.** This sub-module specializes in decoding storage keys and values captured in the EVM trace. By utilizing a dynamic storage tracking mechanism, OpenTracer is able to interpret complex storage structures. It tracks the computations of storage slots through EVM operations like *sha3*, facilitating the decoding of storage accesses in the trace data. This capability enables a deeper understanding of how data is loaded and stored during contract execution. As shown in Box 3 of Figure 1, OpenTracer tracks how a storage slot computed for each `sload` and `sstore` operation, providing a detailed decoded storage access information.

## 3.4 Data Extraction

The data extraction functionality of OpenTracer identifies and analyzes specific trace snippets of interest, such as those accessing a particular storage slot or invoking a specific function. This module supports advanced techniques like taint analysis to trace malicious data flows and data flow analysis to monitor changes across stack, memory, and storage resulting from each EVM instruction. These capabilities are essential for pinpointing potential security vulnerabilities and understanding intricate data flows within smart contracts. An example of invariant-related data extracted, demonstrating the module's output, is depicted in Box 4 of Figure 1. This step ensures the comprehensive collection of data specified by users, effectively meeting diverse analytical needs.

## 3.5 Implementation, Optimization, Use Cases

OpenTracer is developed in Python and consists of 12, 049 lines of code. The architecture allows for optimizations including caching of intermediate results like raw trace data and transaction receipts as illustrated in Figure 1. The system's design also supports parallel processing of tasks such as data downloading, trace parsing, and decoding, significantly enhancing performance and efficiency.

**Use case 1: Invariant Generation.** A fundamental application of OpenTracer is invariant generation for a given contract. Given a contract address as an input, users first obtain its transaction history using tools like TrueBlocks [19]. Then users may either employ existing methodologies or develop other techniques for data collection. Subsequently, OpenTracer is deployed to extract this data from each transaction. This extracted data is then leveraged to generate invariants based on predefined templates, concretizing any undetermined parameters within these templates. When OpenTracer was used in Trace2Inv, it already has 23 different types of invariant templates across 8 categories available for users to choose from. An example of this functionality is illustrated in Box 5 of Figure 1, showcasing a concrete invariant generated from the data extracted for a specific function.

**Use case 2: Trace Translation.** Another key application of OpenTracer is its ability to translate EVM trace data into formats required by other tools, enhancing its utility within the blockchain ecosystem. This capability is pivotal as OpenTracer extracts comprehensive details like function call data, storage accesses, and data flow information for each EVM instruction. Users have the flexibility to devise their own translation methods in the parser or decoder modules, adapting the EVM trace data into the specific formats needed by other tools. For instance, the EVM trace data can be reformatted to meet the requirements of tools like TxSpector [21], effectively substituting the need for an archive node in such scenarios. An example of this translated trace is depicted in Box 6 of Figure 1, showcasing how translated trace data is prepared for use by other analytical tools.

## 4 EVALUATION

In this section, we evaluate the applicability and performance of OpenTracer by applying it to real-world contracts. OpenTracer, utilized in Trace2Inv, efficiently generates 23 invariants across 8 distinct categories. These categories include Access Control, Time Lock, Gas Control, Oracle Slippage, Re-entrancy, and Money Flow, where OpenTracer derives invariants solely from function-level invocation trees. In the "Special Storage" category, OpenTracer employs the contract's storage layout to decode and extract storage data for invariant generation. The "Data Flow" category utilizes both taint and data flow analyses to extract data needed.

As detailed in Table 2, our evaluation encompassed the transaction histories of 42 contracts, known as victim contracts, from their deployment until they got hacked, totaling 350, 800 transactions. Using 70% of these transactions as a training set, OpenTracer successfully generated 659 invariants, averaging 15.69 invariants per contract. Our results demonstrate that for each victim contract, at least one invariant was effective in protecting against its exploit. Notably, the most effective invariant *GasStartUpperBound* alone is capable of protecting 30 out of 42 contracts from their respective exploits. These results underscore the effectiveness of OpenTracer in generating invariants that can safeguard smart contracts from common vulnerabilities.

**Table 2: Summary of Evaluated Contracts and Transactions**

| Metric | Value |
|---|---|
| # Contracts Applied | 42 |
| Total Transactions Analyzed | 350,800 |
| Average Invariants per Contract | 15.69 |
| # Contracts Protected for all invariants | 42 out of 42 |
| # Contracts Protected for the best invariant | 30 out of 42 |

Table 3 showcases the performance of OpenTracer when analyzing the transaction history of the Punk_1 contract, comprising 31 transactions. The test ran on a MacBook Pro with an Apple M2 chip (8 cores, 8 GB RAM).

OpenTracer required a total of 8.55 seconds to parse all 31 transactions. The data collection process for invariant-related information, which involves complex operations such as taint analysis and data flow tracking, generally takes longer. Specifically, the extraction of data for 21 invariants—excluding 2 Oracle Slippage invariants not applicable to Punk_1—averages 3.965 seconds per transaction. The time to extract data ranges from a minimum of 1.125 seconds to a maximum of 5.242 seconds per transaction. These metrics highlight OpenTracer's efficiency in data collection. Additionally, the data collection process is designed to be parallelizable, allowing for simultaneous analysis of multiple transactions, which can significantly enhance efficiency. The translation of all 31 transactions to TxSpector input format is completed in a mere 7.46 seconds, underscoring the swift processing capability of OpenTracer. These performance metrics clearly demonstrate that OpenTracer is not only fast and efficient but also capable of handling large-scale transaction histories effectively.

**Table 3: Performance Metrics of OpenTracer for Punk_1 Transactions**

| Task | Time (s) |
|---|---|
| Parse All Txs | 8.55 |
| Max Data Collection Time Per Tx | 5.242 |
| Min Data Collection Time Per Tx | 1.125 |
| Avg Data Collection Time Per Tx | 3.965 |
| Infer Invariants | 16.58 |
| Translate All Txs to TxSpector Format | 7.46 |

## 5 CONCLUSION

Smart contracts facilitate reliable blockchain transactions without centralized oversight, yet comprehensive tools for dynamic analysis of their transaction histories have been lacking. OpenTracer addresses this gap by providing detailed tracking of every execution step. Available as an open-source tool, OpenTracer offers extensive resources for researchers to explore and validate new invariants, showcasing its effectiveness and efficiency. Its robust performance and accessibility position OpenTracer as a pivotal resource in advancing the security of smart contracts.

## REFERENCES


[1] 2018. Solidity. https://solidity.readthedocs.io/en/v0.5.1/.
[2] 2018. Vyper. https://docs.vyperlang.org/en/stable/.
[3] 2020. Etherscan. https://etherscan.io.
[4] alchemy. 2024. alchemy. https://www.alchemy.com/.
[5] BlockSec. 2024. Phalcon. https://app.blocksec.com/explorer/.
[6] Ting Chen, Rong Cao, Ting Li, Xiapu Luo, Guofei Gu, Yufei Zhang, Zhou Liao, Hang Zhu, Gang Chen, Zheyuan He, et al. 2020. SODA: A Generic Online Detection Framework for Smart Contracts.. In *NDSS*.
[7] Zhiyang Chen, Ye Liu, Sidi Mohamed Beillahi, Yi Li, and Fan Long. 2024. Demystifying Invariant Effectiveness for Securing Smart Contracts. *arXiv preprint arXiv:2404.14580* (2024).
[8] Ye Liu and Yi Li. 2022. InvCon: A Dynamic Invariant Detector for Ethereum Smart Contracts. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–4.
[9] Ye Liu, Yi Li, Shang-Wei Lin, and Cyrille Artho. 2022. Finding permission bugs in smart contracts with role mining. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*. 716–727.
[10] Ye Liu, Chengxuan Zhang, et al. 2024. Automated Invariant Generation for Solidity Smart Contracts. *arXiv preprint arXiv:2401.00650* (2024).
[11] OpenChain. 2024. OpenChain. https://openchain.xyz/trace.
[12] QuickNode. 2024. QuickNode. hhttps://www.quicknode.com/.
[13] Michael Rodler, Wenting Li, Ghassan O Karame, and Lucas Davi. 2018. Sereum: Protecting existing smart contracts against re-entrancy attacks. *arXiv preprint arXiv:1812.05934* (2018).
[14] Tenderly. 2024. Tenderly. https://dashboard.tenderly.co/explorer.
[15] TokenFlow. 2024. AnyTx. https://app.tokenflow.live/anytx.
[16] TokenFlow. 2024. EthTx. https://ethtx.info/.
[17] TokenFlow. 2024. EthTx Source. https://github.com/EthTx/ethtx/tree/master.
[18] Christof Ferreira Torres, Antonio Ken Iannillo, Arthur Gervais, and Radu State. 2021. The Eye of Horus: Spotting and Analyzing Attacks on Ethereum Smart Contracts. In *Financial Cryptography and Data Security - 25th International Conference, FC 2021, Virtual Event, March 1-5, 2021, Revised Selected Papers, Part I (Lecture Notes in Computer Science, Vol. 12674)*, Nikita Borisov and Claudia Díaz (Eds.). Springer, 33–52. https://doi.org/10.1007/978-3-662-64322-8_2
[19] TrueBlocks Team. 2022. *TrueBlocks: Lightweight indexing for any EVM-based blockchain*. https://trueblocks.io/
[20] Siwei Wu, Lei Wu, Yajin Zhou, Runhuai Li, Zhi Wang, Xiapu Luo, Cong Wang, and Kui Ren. 2022. Time-travel Investigation: Toward Building a Scalable Attack Detection Framework on Ethereum. *ACM Trans. Softw. Eng. Methodol.* 31, 3 (2022), 54:1–54:33. https://doi.org/10.1145/3505263
[21] Mengya Zhang, Xiaokuan Zhang, Yinqian Zhang, and Zhiqiang Lin. 2020. TXSPECTOR: Uncovering Attacks in Ethereum from Transactions. In *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, Srdjan Capkun and Franziska Roesner (Eds.). USENIX Association, 2775–2792. https://www.usenix.org/conference/usenixsecurity20/presentation/zhang-mengya