**Code Analysis Tools for Solidity Smart Contracts: An Overview**

By

[Lekshmi Pg](https://medium.com/u/ba95ad71cccb?source=post_page-----b3642939c195--------------------------------)

, Research Fellow,

[Kerala Blockchain Academy](https://medium.com/u/d86fe338e094?source=post_page-----b3642939c195--------------------------------)

Software developers may make mistakes perpetually. These mistakes may range from a missing semicolon, or a mismatch in parenthesis, to logical errors in the program. While the compiler notifies syntactical errors, some other errors go unnoticed. In a typical software development cycle, software revisions can fix bugs discovered after the software release.



As you are already aware, smart contracts are immutable. Once deployed on the blockchain, it will remain entrenched till the blockchain is live. This makes code analysis a critical part of smart contract development.

**What is Code Analysis?**

Code analysis examines software code to identify potential problems or errors, verify that it adheres to coding standards, and ensure that it is high-quality. Code analysis can be performed either statically by examining the code without executing it or dynamically by executing it and analyzing its behavior while running.

There are many tools and techniques available for performing code analysis. They can be broadly classified into static and dynamic code analysis tools.

**Static code analysis is**used to identify many common coding problems before a program is released. It involves examining the code manually or using tools to automate the process. Static code analysis tools can automatically scan the code without executing it. They can identify syntax errors, check for adherence to coding standards, and look for potential security vulnerabilities. It is generally performed early in the development process before the code is deployed. It can help identify problems that may be difficult to locate during testing and save time and resources by detecting issues before they become more serious.

**Dynamic code analysis** involves executing the code and analyzing its behavior while running. They can identify runtime errors, test the code’s performance, and verify that it behaves as intended. It is typically performed after the code has been deployed. It can identify issues that may not have been apparent during testing, such as performance bottlenecks or security vulnerabilities that only occur under specific conditions.

Let me introduce some popular smart contract analysis tools for solidity smart contracts.

**Slither**

Slither is an open-source smart contract analysis tool that statically analyses the code. Slither takes Solidity Abstract Syntax Tree(AST) developed by Solidity Compiler from the source code as input. Slither tool consists of a set of detectors to detect the vulnerabilities in the contract. It also provides detailed visual information about the contract, and also it will provide suggestions to optimize the code. Slither provides an API for interaction.

*Abstract Syntax Tree is a data structure widely used by compilers to represent the program structure. AST is a tree structure that represents the source code in formal language. Each node of the tree is a construct in the code.*

Some of the vulnerabilities detected by Slither include

* [Incorrect ABI encoder usage](https://github.com/crytic/slither/wiki/Detector-Documentation#storage-abiencoderv2-array)
* [Arbitrary-send-erc20](https://github.com/crytic/slither/wiki/Detector-Documentation#arbitrary-from-in-transferfrom)
* [Modifying storage array by value](https://github.com/crytic/slither/wiki/Detector-Documentation#modifying-storage-array-by-value)
* [Incorrect-shift](https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-shift-in-assembly)
* [Multiple constructor scheme](https://github.com/crytic/slither/wiki/Detector-Documentation#multiple-constructor-schemes)
* [Multiple contracts with the same name](https://github.com/crytic/slither/wiki/Detector-Documentation#name-reused)
* [Detect unprotected variables](https://github.com/crytic/slither/wiki/Detector-Documentation#protected-variables)
* [Public mappings with nested variables](https://github.com/crytic/slither/wiki/Detector-Documentation#public-mappings-with-nested-variables)
* [State variable shadowing](https://github.com/crytic/slither/wiki/Detector-Documentation#state-variable-shadowing)
* [Unprotected call to self-destruct or suicide](https://github.com/crytic/slither/wiki/Detector-Documentation#suicidal)
* [Uninitialized state variables](https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-state-variables)
* [Uninitialized storage variables](https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-storage-variables)
* [Unprotected upgradeable contract](https://github.com/crytic/slither/wiki/Detector-Documentation#unprotected-upgradeable-contract)
* [Sending Ether to an arbitrary address](https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations)

Slither provides us with a human-readable summary of the contract. It also identifies the contracts inherited by a specific contract and represents it using a graph. Slither is written in python, so it needs Python 3 for installation and solc.

Though it claims to be effective, false positives and negatives（假阳性和假阴性） occur in certain instances.

*Analysis tools can have both false positives and false negatives. False positive refers to a case where the tool reports an issue that is not necessarily an error. False negatives are undetected errors. Some related studies also point out that many of the defects can be detected by static analysis tools in theory, but are not detected in practice due to certain limitations of the tools.*

**Mythril**

Mythril is a Smart contract analysis tool for solidity developed in python by ConsenSys. It uses symbolic execution to analyze the source code. Mythril will analyze the code by executing the code and monitoring its behavior on different inputs.

Mythril detection capabilities include:

* [Delegate Call To Untrusted Contract](https://mythril-classic.readthedocs.io/en/master/module-list.html#delegate-call-to-untrusted-contract)
* [Dependence on Predictable Variables](https://mythril-classic.readthedocs.io/en/master/module-list.html#dependence-on-predictable-variables)
* [Unprotected Ether Withdrawal](https://mythril-classic.readthedocs.io/en/master/module-list.html#ether-thief)
* [Assert Violation](https://mythril-classic.readthedocs.io/en/master/module-list.html#exceptions)
* [Reentrancy](https://mythril-classic.readthedocs.io/en/master/module-list.html#external-calls)
* [Integer Overflow and Underflow](https://mythril-classic.readthedocs.io/en/master/module-list.html#integer)
* [DoS with Failed Call](https://mythril-classic.readthedocs.io/en/master/module-list.html#multiple-sends)
* [Unprotected SELF-DESTRUCT Instruction](https://mythril-classic.readthedocs.io/en/master/module-list.html#suicide)
* [Unchecked Call Return Value](https://mythril-classic.readthedocs.io/en/master/module-list.html#user-supplied-assertion)
* [Write to Arbitrary Storage Location](https://mythril-classic.readthedocs.io/en/master/module-list.html#arbitrary-storage-write)
* [Arbitrary Jump with Function Type Variable](https://mythril-classic.readthedocs.io/en/master/module-list.html#arbitrary-jump)

If Mythril finds any issues with the contract, it will generate a report detailing the problems it has identified and rate the vulnerabilities based on their level of danger. It is also used (in combination with other tools and techniques) in the [MythX](https://mythx.io/" \t "_blank) security analysis platform.

**Echidna**

Echidna is designed for fuzzing/property-based testing and developed in Haskell programming language. Fuzzing is a technique that includes generating random inputs and makes it easy to automatically generate tests to detect violations in assertions and custom properties.

Actually, Echidna does their analysis based on the projects. In Echidna, the first step for analysis is to generate invariants or test cases that are written as solidity functions based on the source code and the input/output behavior. Then these invariants are executed with random transactions and then reviewed the test results. Since analysis is done based on the invariants specific to the project, there is no chance for false positives. If Echidna finds any issues with the contract, it will generate a report detailing the problems it has identified, along with information about the test cases that triggered the issues.

**Securify v2.0**

This is another static analysis tool for solidity smart contract analysis supported by Ethereum Foundation and ChainSecurity. It takes EVM Bytecode generated by the Solidity compiler as input. It can detect vulnerabilities present in EVM bytecode. Securify can analyze up to 37 vulnerabilities. Securify supports only flat contracts; it doesn’t support contracts with import statements.

Some of the vulnerabilities identified by Securify include the following:

* [Unexpected Ether Balance](https://swcregistry.io/docs/SWC-107)
* [Reentrancy](https://swcregistry.io/docs/SWC-107)
* [Authorization through tx.origin](https://swcregistry.io/docs/SWC-115)
* [Unchecked send](https://swcregistry.io/docs/SWC-105)
* [Unused return value](https://swcregistry.io/docs/SWC-104)
* [Block values for timestamp](https://swcregistry.io/docs/SWC-116)
* [Floating Pragma](https://swcregistry.io/docs/SWC-103)

If Securify finds any problems, it will generate a report detailing the issues it has identified, along with recommendations for how to fix them. It may also include information about the contract’s structure and dependencies and a summary of its behavior.

**RemixIDE static code analysis plugin**

This tool is a plugin for Remix IDE, the online solidity coding platform. It is available both as a browser version and a desktop version.

Some of the vulnerabilities detected by Remix IDE include:

* [Reentrancy](https://remix-ide.readthedocs.io/en/latest/static_analysis.html)
* [Authorization through tx.origin](https://remix-ide.readthedocs.io/en/latest/static_analysis.html)
* [Low-level calls](https://remix-ide.readthedocs.io/en/latest/static_analysis.html)
* [Self-destruct](https://remix-ide.readthedocs.io/en/latest/static_analysis.html)
* [In-line assembly use](https://remix-ide.readthedocs.io/en/latest/static_analysis.html)

It also detects gas-consumption-related vulnerabilities and also vulnerabilities based on code optimization. Altogether it has around 21 analysis modules grouped into four categories: *Security*, *Gas* & *Economy*, *ERC*, and *Miscellaneous*. If the static analysis tool finds any issues with the contract, it will generate a report detailing the problems it has identified, along with recommendations for how to fix them.

***Wrapping up***

*Smart Contract analysis is a crucial activity since smart contracts are immutable. Once deployed, it can’t be changed. The errors and security vulnerabilities will persist if present in a smart contract. Smart contracts pass through multiple levels of scrutiny and auditing before their final deployment.*

*We can use the combination of static and dynamic analysis tools to effectively detect vulnerabilities in smart contracts. In such cases, the static analysis tool detects the vulnerabilities present in the code before execution. Generally, it deals with the bytecode format generated by the compiler. It may miss some vulnerabilities that may occur at run time, which can be detected by the dynamic analysis tool.*

*I hope this article helped to familiarize some standard tools used for smart contract code analysis. Each tool has its own strengths and weaknesses. The choice of a tool normally depends on the category of smart contract and possible error scenarios. We will come up with another article familiarizing some more tools.*

*Reference*

[1] <https://github.com/crytic/slither>

[2] <https://github.com/ConsenSys/mythril>

[3] <https://github.com/crytic/echidna>

[4] <https://github.com/eth-sri/securify2>

[5] <https://github.com/smartdec/smartcheck>

[6] <https://remix-ide.readthedocs.io/en/latest/static_analysis.html>

[7] Nipun Sharma, Swati Sharma,” A Survey of Mythril, A Smart Contract Security Analysis Tool for EVM Bytecode”, December 2022

[8] Josselin Feist, Alex Groce, ”Slither: A Static Analysis Framework For Smart Contracts”, August 2019

*Image Courtesy*: [Freepik](https://www.freepik.com/" \t "_blank)