CERTIK

CERTIK SECURITY TOOLING & RESOURCE GUIDE

For Smart Contract Developers

CertiK Security Tooling & Resource Guide

For Smart Contract Developers

Smart contracts have revolutionized the way we perceive and interact with programmable logic in the context of blockchain technology. With the advent of platforms like Ethereum, we've seen an explosive growth in decentralized applications (DApps) powered by these self-executing contracts. However, the unique properties of smart contracts, such as their immutability and direct control over digital assets, make them an attractive target for attackers. The security of smart contracts thus becomes a paramount concern, necessitating a robust suite of security tooling and resources for developers.

This report aims to provide a comprehensive overview and guide on essential security tools and resources that are invaluable for smart contract developers. Our objective is to ensure that developers have a holistic understanding of the tools at their disposal, how to effectively leverage them, and how they interplay in the wider security ecosystem.

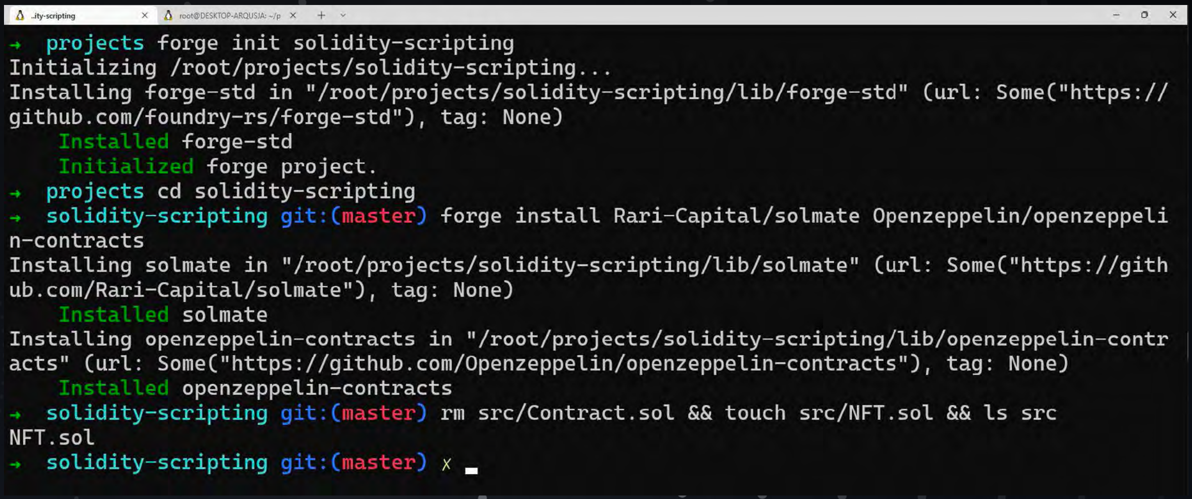
Each tool and resource is examined in-depth, with a focus on its practical applications, strengths, and potential limitations. By bringing these resources together in a single guide, we hope to aid smart contract developers in fortifying their code, improving their workflows, and ultimately contributing to a more secure and robust decentralized future.

Tools and Resources:

- Foundry
- VS Code
- Solidity Visual Developer
- Tenderly
- SWC Registry
- EVM Codes
- GitHub Compare
- UnixTime

Foundry

Foundry is a smart contract development tool that helps developers manage dependencies, compile projects, run tests, deploy contracts, & interact with the Ethereum blockchain from the command line.



```
→  projects forge init solidity-scripting
Initializing /root/projects/solidity-scripting...
Installing forge-std in "/root/projects/solidity-scripting/lib/forge-std" (url: Some("https://
github.com/foundry-rs/forge-std"), tag: None)
    Installed forge-std
    Initialized forge project.
→  projects cd solidity-scripting
→  solidity-scripting git:(master) forge install Rari-Capital/solmate Openzeppelin/openzeppeli
n-contracts
Installing solmate in "/root/projects/solidity-scripting/lib/solmate" (url: Some("https://gith
ub.com/Rari-Capital/solmate"), tag: None)
    Installed solmate
Installing openzeppelin-contracts in "/root/projects/solidity-scripting/lib/openzeppelin-contr
acts" (url: Some("https://github.com/Openzeppelin/openzeppelin-contracts"), tag: None)
    Installed openzeppelin-contracts
→  solidity-scripting git:(master) rm src/Contract.sol && touch src/NFT.sol && ls src
NFT.sol
→  solidity-scripting git:(master) ✗ ▬
```

Foundry is built on Rust and provides several tools and features to facilitate the development, testing, and deployment of smart contracts. It provides tools for gas tracking, debugging, and continuous integration. Foundry supports Solidity, the primary language for writing smart contracts on Ethereum.

One of the key features of Foundry is its testing framework, Forge. It provides fuzz testing, a code auditing method that involves feeding a series of random inputs to your program to identify any weaknesses or faults. Fuzz testing is particularly valuable in a blockchain context because once a smart contract is deployed, it cannot be changed. It's critical to ensure that it behaves correctly with a wide range of inputs before it goes live.

Foundry also includes tools named Cast, Anvil, and Chisel.

Cast is Foundry's command line interface for performing Ethereum RPC calls. It provides a range of commands that allow developers to easily query the blockchain, send transactions, estimate gas, and more.

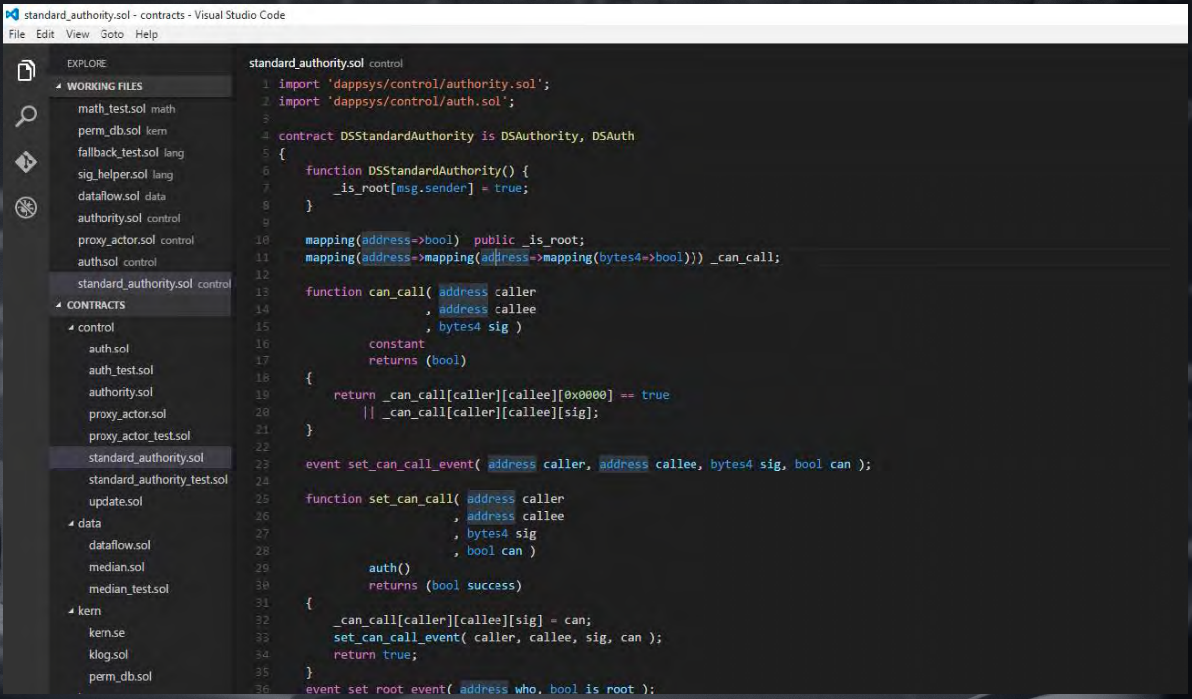
Anvil is a local testnet tool that helps with the deployment and verification of contracts. It also provides features for creating and managing development networks.

Chisel is a Solidity REPL (“Read-Eval-Print Loop”) that allows developers to write and test Solidity code snippets. The tool offers verbose feedback on each input. It can be used to quickly test the behavior of Solidity snippets on a local or forked network. Chisel is a valuable tool for rapidly testing and experimenting with Solidity code, eliminating the need to set up a sandbox test suite.

VS Code

Visual Studio Code (VS Code) is a source code editor developed by Microsoft.

It is free, open-source, and runs on multiple platforms, including Windows, Linux, and macOS. It provides support for debugging, embedded Git control, syntax highlighting, intelligent code completion, snippets, and code refactoring. It also supports a wide variety of programming languages, which can be extended through its robust plugin system.



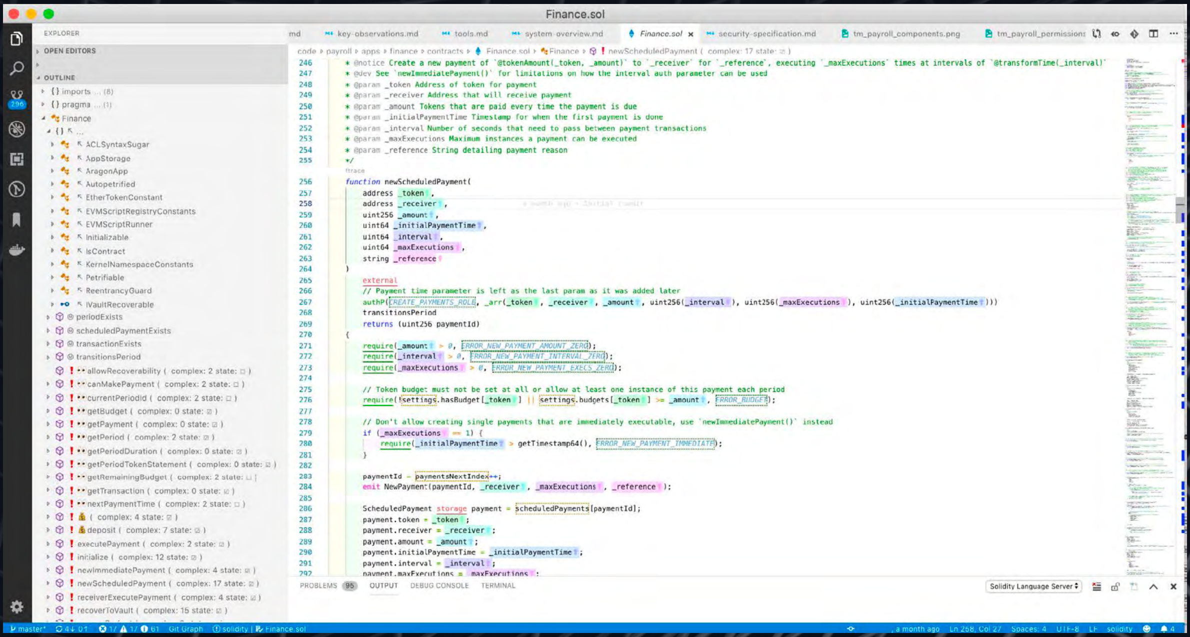
```
import 'dappsys/control/authority.sol';
import 'dappsys/control/auth.sol';

contract DSStandardAuthority is DSAuthority, DSAuth
{
    function DSStandardAuthority() {
        _is_root[msg.sender] = true;
    }

    mapping(address=>bool)  public _is_root;
    mapping(address=>mapping(address=>mapping(bytes4=>bool))) _can_call;

    function can_call( address caller
                     , address callee
                     , bytes4 sig )
             constant
             returns (bool)
    {
        return _can_call[caller][callee][0x0000] == true
            || _can_call[caller][callee][sig];
    }

    event set_can_call_event( address caller, address callee, bytes4 sig, bool can );

    function set_can_call( address caller
                         , address callee
                         , bytes4 sig
                         , bool can )
             auth()
             returns (bool success)
    {
        _can_call[caller][callee][sig] = can;
        set_can_call_event( caller, callee, sig, can );
        return true;
    }
    event set_root_event( address who, bool is_root );
```

Some powerful features of VS Code include:

- **Syntax Highlighting and Code Formatting:** This makes reading and understanding the code easier. Smart contract auditors and developers need to understand every line of code to ensure there are no security vulnerabilities. Syntax highlighting and code formatting can be especially useful when dealing with languages like Solidity, which is commonly used for writing Ethereum smart contracts.
- **Debugging Tools:** VS Code offers integrated debugging tools, which can be very helpful for auditors. They can step through the code, inspect variables, and view call stacks. This makes it easier to identify and understand the behavior of potentially harmful code.
- **Integrated Terminal:** VS Code features an integrated terminal that can be used for executing script commands, running tests, and other tasks. This can streamline the workflow of an auditor.
- **Version Control:** With embedded Git control, auditors can easily track changes, view differences between versions, and ensure that no code has been altered without proper review.
- **Extensions:** There are many available extensions for VS Code that can further help in auditing smart contracts. For instance, there are linters for Solidity that can automatically flag potential issues in the code. There are also extensions that provide features like code navigation and automatic compilation of contracts.
- **Multi-platform and Lightweight:** VS Code runs on multiple operating systems and is less resource-intensive compared to full-featured IDEs. This means auditors can use it on their preferred hardware and software setups.
- **Collaboration Features:** With the “Live Share” extension, multiple auditors can work on the same codebase in real time. This can be very beneficial for larger projects that require collaboration between multiple auditors.

Solidity Visual Developer

Solidity Visual Developer offers a range of features that can assist developers in writing and reviewing Solidity code.



```
 * @notice Create a new payment of `@tokenAmount(_token, _amount)` to `_receiver` for `_reference`, executing `_maxExecutions` times at intervals of `@transformTime(_interval)`
 * @dev See `newImmediatePayment()` for limitations on how the interval auth parameter can be used
 * @param _token Address of token for payment
 * @param _receiver Address that will receive payment
 * @param _amount Tokens that are paid every time the payment is due
 * @param _initialPaymentTime Timestamp for when the first payment is done
 * @param _interval Number of seconds that need to pass between payment transactions
 * @param _maxExecutions Maximum instances a payment can be executed
 * @param _reference String detailing payment reason
 */
function newScheduledPayment(
    address _token,
    address _receiver,
    uint256 _amount,
    uint64 _initialPaymentTime,
    uint64 _interval,
    uint64 _maxExecutions,
    string _reference
)
    external
    // Payment time parameter is left as the last param as it was added later
    authP(CREATE_PAYMENTS_ROLE, _arr(_token, _receiver, _amount, uint256(_interval), uint256(_maxExecutions), uint256(_initialPaymentTime)))
    transitionsPeriod
    returns (uint256 paymentId)
{
    require(_amount > 0, ERROR_NEW_PAYMENT_AMOUNT_ZERO);
    require(_interval > 0, ERROR_NEW_PAYMENT_INTERVAL_ZERO);
    require(_maxExecutions > 0, ERROR_NEW_PAYMENT_EXECS_ZERO);

    // Token budget must not be set at all or allow at least one instance of this payment each period
    require(!settings.hasBudget[_token] || settings.budgets[_token] >= _amount, ERROR_BUDGET);

    // Don't allow creating single payments that are immediately executable, use `newImmediatePayment()` instead
    if (_maxExecutions == 1) {
        require(_initialPaymentTime > getTimestamp64(), ERROR_NEW_PAYMENT_IMMEDIATE);
    }

    paymentId = paymentsNextIndex++;
    emit NewPayment(paymentId, _receiver, _maxExecutions, _reference);

    ScheduledPayment storage payment = scheduledPayments[paymentId];
    payment.token = _token;
    payment.receiver = _receiver;
    payment.amount = _amount;
    payment.initialPaymentTime = _initialPaymentTime;
    payment.interval = _interval;
    payment.maxExecutions = _maxExecutions;
```

It can be particularly useful for smart contract auditors by providing enhanced security-related syntax highlighting, advanced Solidity insights & code augmentation features that draw attention to security-critical aspects of the code.

Some of the features offered by Solidity Visual Developer include:

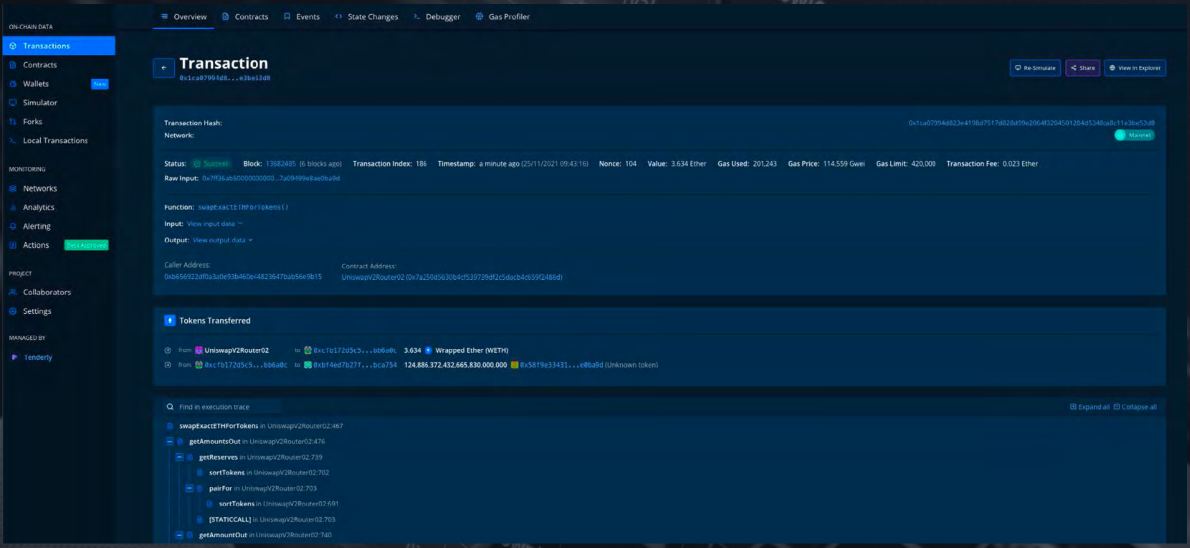
- **Semantic Highlighting:** This feature highlights state variables, detects and alerts about state variable shadowing, and highlights function arguments in the function body.
- **Review Features:** This includes audit annotations or bookmarks, importation of external scanner results, and several codelens inline actions like graph, report, dependencies, inheritance, parse, ftrace, flatten, generate unit test stub, function signature hashes, and more.
- **Graph and Reporting Features:** SVD allows you to generate interactive call

graphs with call flow highlighting and automatically generate UML diagrams from code to support threat modeling exercises or documentation.

- **Code Augmentation:** When you hover over Ethereum Account addresses, the extension allows you to download the byte-code, source-code, or open it in the browser. It also provides tooltips for ASM instructions, keywords, and StateVar's declarations.
- **Views:** The Cockpit View and Outline View provide a comprehensive layout of your Solidity files, including contracts, stateVars, methods, inherited names, and security relevant information. It also calculates complexity rating and annotates functions with information about whether they are accessing stateVars.
- **Ethereum Account Address Actions:** It enables actions like opening the account on etherscan.io, showing the contract code, showing the verified contract source code, and decompiling the bytecode.
- **Semantic Function Argument Highlighting:** Arguments are assigned different colors in the scope of the function for easier reading and understanding.
- **Inline Bookmarks:** Allows you to flag lines for security review or start a security review discussion.
- **Code Augmentation / Annotations / Hover / Tooltip:** Provides additional information for various keywords (including security notes), assembly instruction signatures, address hover integration, and more.
- **State Variable Highlighting:** Highlights local state variables, alerts on a shadowed variable, highlights constant state variables, and highlights inherited state variables.
- **CodeLenses:** It provides interactive graphs, generates reports, shows inheritance, shows AST, allows flattening source files, provides ftrace, auto-generates UML for source-units or specific contracts, and shows function signature hashes.
- **Outline View:** It shows the library with function parameters and declarations, classes, events, and functions annotated (stateMutability, visibility) and declarations.

Tenderly

Tenderly is a platform designed to help Ethereum developers monitor and troubleshoot their smart contracts. It provides several tools that make it easier to develop, test, and deploy smart contracts on the Ethereum blockchain.



Transaction
0x1ca07994d8...e3be53d8

Transaction Hash: 0x1ca07994d823e4198d7517d828d99e2064f3204501284d5348ca8c11e3be53d8
Network: Mainnet

Status: Success Block: 13682485 (6 blocks ago) Transaction Index: 186 Timestamp: a minute ago (25/11/2021 09:43:16) Nonce: 104 Value: 3.634 Ether Gas Used: 201,243 Gas Price: 114.559 Gwei Gas Limit: 420,000 Transaction Fee: 0.023 Ether
Raw Input: 0x7ff36ab50000000000...7a09499e8ae0ba9d

Function: swapExactETHForTokens()
Input: View input data
Output: View output data

Caller Address: 0xb656922df0a3a0e93b460e44823647bab56e9b15
Contract Address: UniswapV2Router02 (0x7a250d5630b4cf539739df2c5dacb4c659f2488d)

Tokens Transferred

from UniswapV2Router02 to 0xcfb172d5c5...bb6a0c 3.634 Wrapped Ether (WETH)
from 0xcfb172d5c5...bb6a0c to 0xbf4ed7b27f...bca754 124,886,372,432,665,830,000,000 0x58f9e33431...e0ba9d (Unknown token)

```
swapExactETHForTokens in UniswapV2Router02:467
  getAmountsOut in UniswapV2Router02:476
    getReserves in UniswapV2Router02:739
      sortTokens in UniswapV2Router02:702
      pairFor in UniswapV2Router02:703
        sortTokens in UniswapV2Router02:691
      [STATICCALL] in UniswapV2Router02:703
    getAmountOut in UniswapV2Router02:740
```

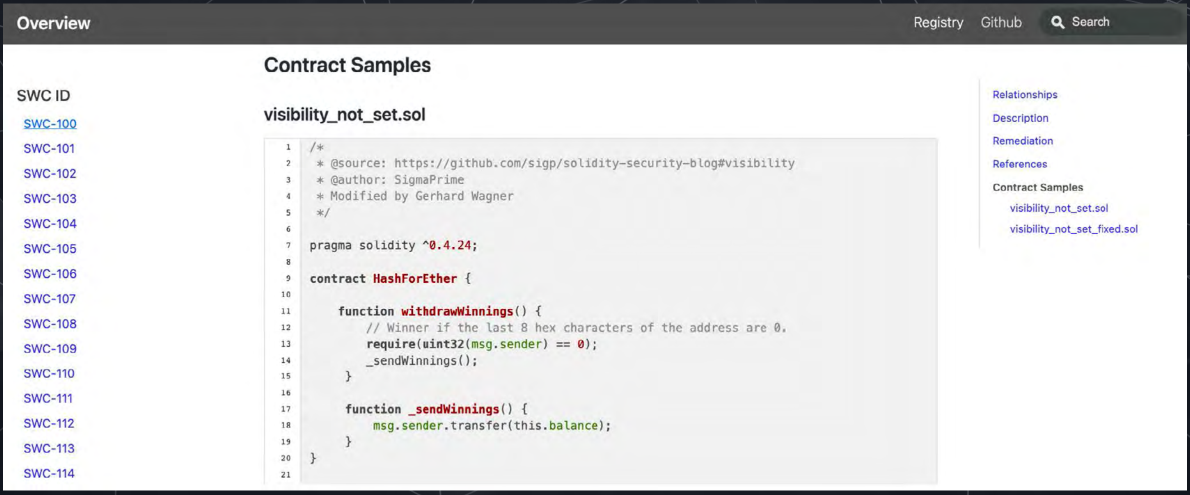
Key features of Tenderly include:

- **Real-time Monitoring and Alerting:** Tenderly allows developers to set up alerts for their smart contracts based on a number of different conditions. This could include things like transaction failures, gas usage, function calls, or other on-chain events.
- **Visual Debugger:** Tenderly offers a visual debugger for Ethereum transactions. This can be used to inspect transactions, view stack traces, and understand exactly why a transaction may have failed.
- **Simulation and Gas Profiler:** Developers can simulate transactions on Tenderly before they're sent to the blockchain. This can help to test changes and predict how they'll affect the contract's performance and gas usage.

- **Multi-Sig Contract Support:** Tenderly supports multi-sig contracts, allowing teams to collaborate on contract development and deployment.
- **Forking Support:** Developers can create a fork of the Ethereum blockchain within Tenderly, allowing them to experiment and test their contracts in a sandboxed environment.
- **Network and Contract Analytics:** Tenderly provides detailed analytics about contract usage and network conditions. This can help developers understand how their contracts are being used and how they can optimize them for better performance.

Tenderly supports both public and private networks, and it can be used with most Ethereum development frameworks, making it a versatile tool for Ethereum developers and smart contract auditors.

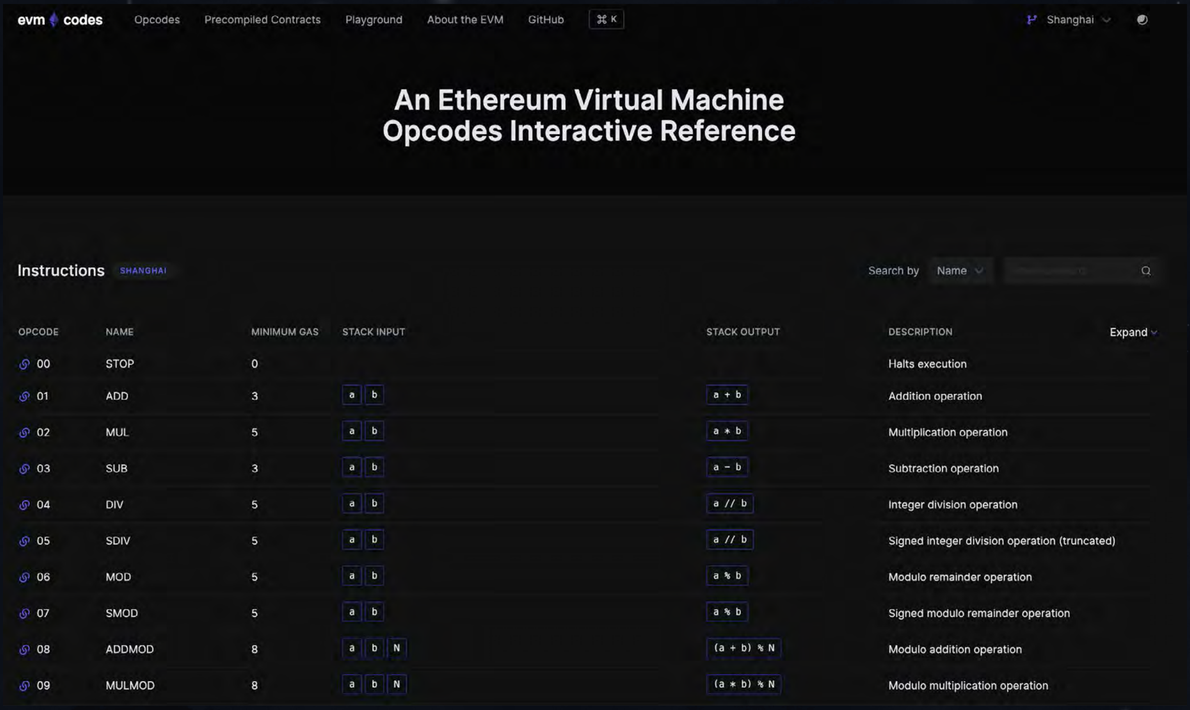
SWC Registry



```
/*
 * @source: https://github.com/sigp/solidity-security-blog#visibility
 * @author: SigmaPrime
 * Modified by Gerhard Wagner
 */

pragma solidity ^0.4.24;

contract HashForEther {

    function withdrawWinnings() {
        // Winner if the last 8 hex characters of the address are 0.
        require(uint32(msg.sender) == 0);
        _sendWinnings();
     }

     function _sendWinnings() {
         msg.sender.transfer(this.balance);
     }
}
```

The Smart Contract Weakness Classification Registry, or SWC Registry, is a realization of the vulnerability classification system suggested in EIP-1470. It is informally structured in harmony with the terms and framework of the Common Weakness Enumeration (CWE), and it integrates an extensive variety of weakness variations that are unique to smart contracts. It includes details of the vulnerability, code examples, as well as remediations.

EVM Codes



An Ethereum Virtual Machine
Opcodes Interactive Reference

Instructions SHANGHAI

OPCODE	NAME	MINIMUM GAS	STACK INPUT	STACK OUTPUT	DESCRIPTION
00	STOP	0			Halts execution
01	ADD	3	a b	a + b	Addition operation
02	MUL	5	a b	a * b	Multiplication operation
03	SUB	3	a b	a - b	Subtraction operation
04	DIV	5	a b	a // b	Integer division operation
05	SDIV	5	a b	a // b	Signed integer division operation (truncated)
06	MOD	5	a b	a % b	Modulo remainder operation
07	SMOD	5	a b	a % b	Signed modulo remainder operation
08	ADDMOD	8	a b N	(a + b) % N	Modulo addition operation
09	MULMOD	8	a b N	(a * b) % N	Modulo multiplication operation

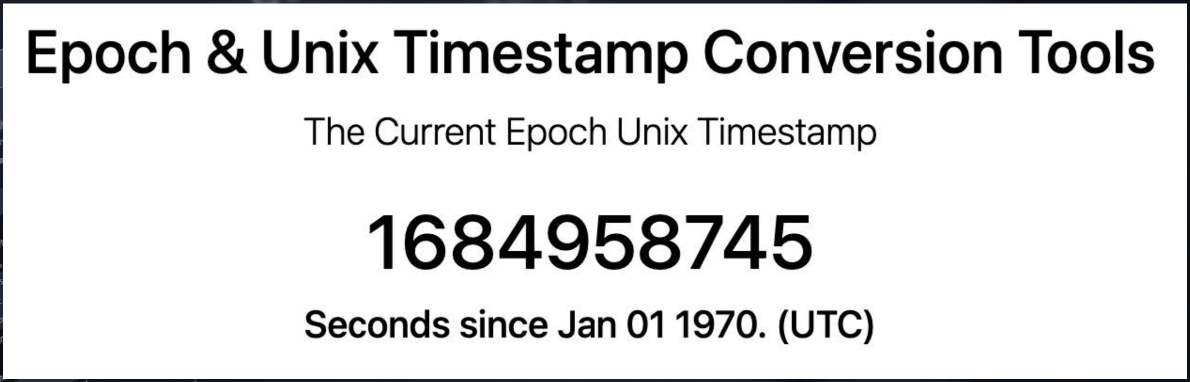
EVM Codes is a searchable registry of different opcodes. Each opcode performs a specific operation within the EVM, such as arithmetic operations (like addition and multiplication), data manipulation (like getting and setting values), control flow (like jumps and conditional jumps), and operations related to Ethereum itself (like getting the balance of an account, creating a new contract, calling another contract, etc.).

Understanding these opcodes can provide insight into how smart contracts are executed at the lowest level. This can help developers optimize their code for gas efficiency and performance.

When smart contracts are compiled, they are translated into opcodes. Understanding the function of opcodes is invaluable when trying to debug issues or perform security analyses on smart contracts. Developers can manually inspect the compiled code, and security auditors can look for potentially dangerous opcodes that could indicate vulnerabilities.

In some cases, developers might only have access to the compiled bytecode of a smart contract (for example, for a contract already deployed on the blockchain). Understanding opcodes can help reverse engineer the contract's functionality.

UnixTime



Epoch & Unix Timestamp Conversion Tools

The Current Epoch Unix Timestamp

1684958745

Seconds since Jan 01 1970. (UTC)

Unixtime.org is a web-based tool for converting Unix timestamps to human-readable date-time formats and vice versa. Unix time, also known as Epoch time, POSIX time, or UNIX Epoch time, is a system for describing a point in time. It counts the number of seconds that have elapsed since the Unix epoch, which is defined as 00:00:00 UTC on 1 January 1970. Unixtime.org supports Unix timestamps in seconds, milliseconds, microseconds, and nanoseconds. It also provides conversion functionality to various time formats. This simple tool makes it easy to get a Unix timestamp for use in testing.

Github Compare

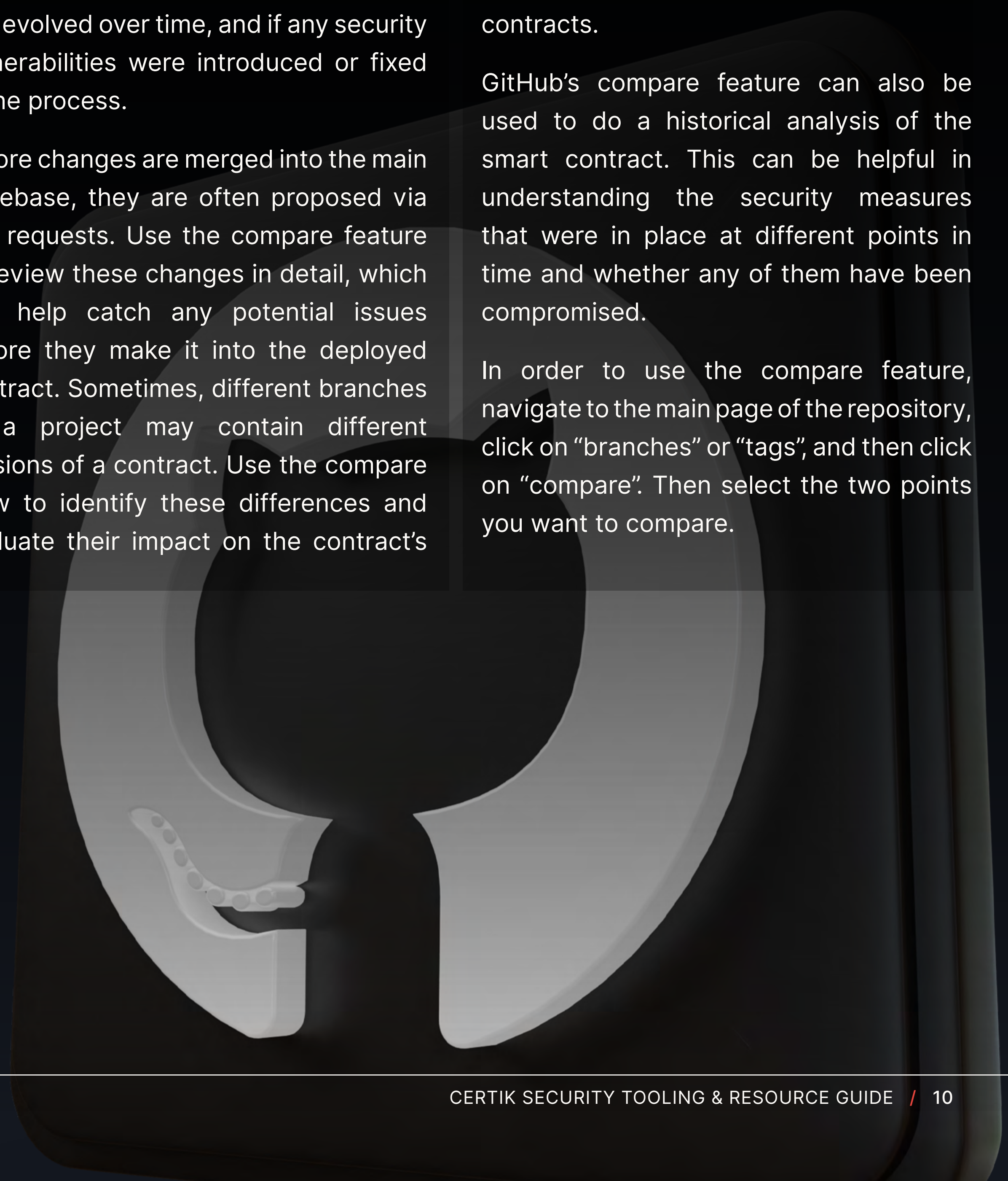
GitHub's compare view allows users to compare the changes between two points in time within the same repository.

This feature can be used to compare different versions of a smart contract's code. This can show how the contract has evolved over time, and if any security vulnerabilities were introduced or fixed in the process.

Before changes are merged into the main codebase, they are often proposed via pull requests. Use the compare feature to review these changes in detail, which can help catch any potential issues before they make it into the deployed contract. Sometimes, different branches of a project may contain different versions of a contract. Use the compare view to identify these differences and evaluate their impact on the contract's security. If there are other contracts known to be secure, compare the contract you're auditing with these known-secure contracts.

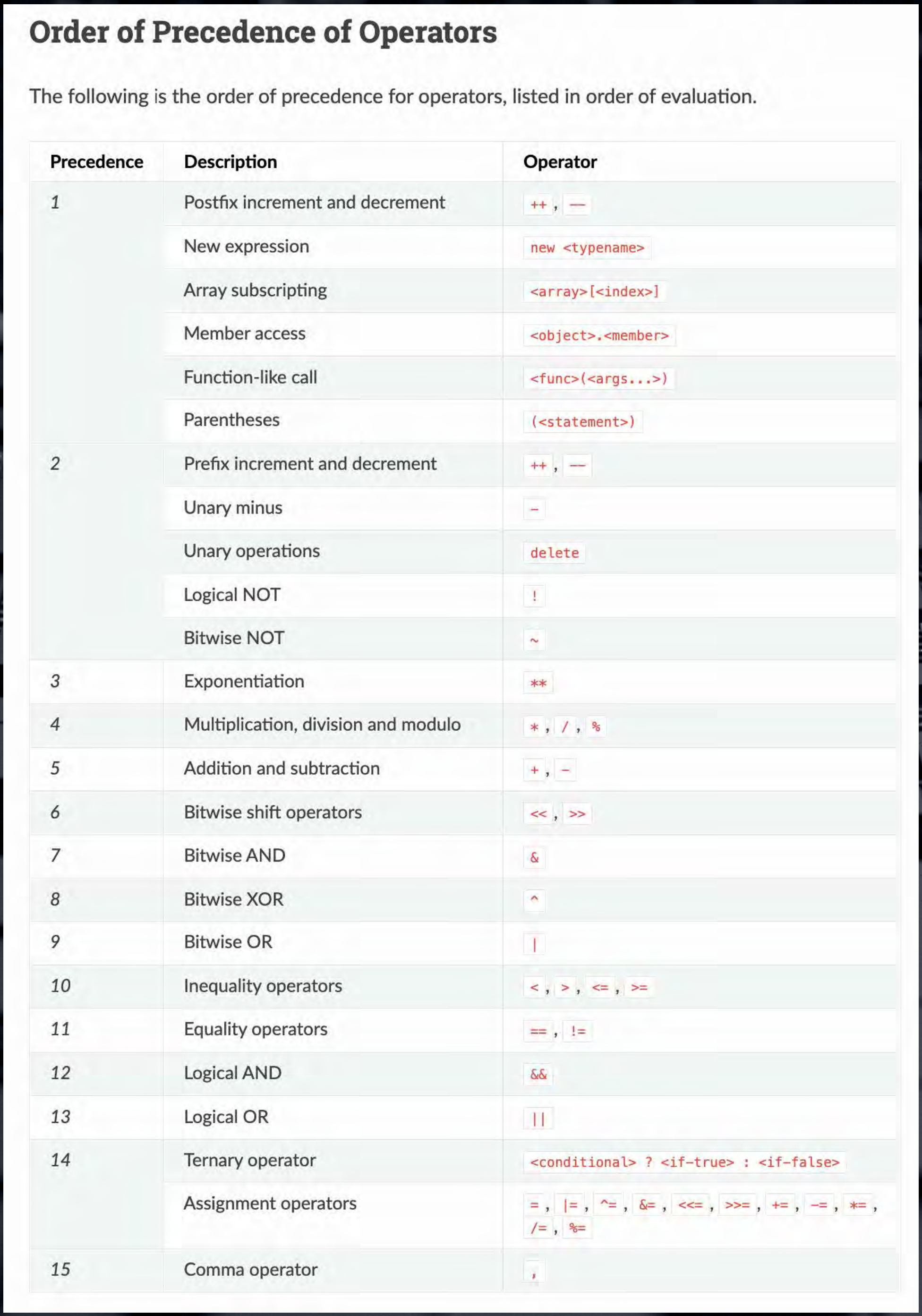
GitHub's compare feature can also be used to do a historical analysis of the smart contract. This can be helpful in understanding the security measures that were in place at different points in time and whether any of them have been compromised.

In order to use the compare feature, navigate to the main page of the repository, click on "branches" or "tags", and then click on "compare". Then select the two points you want to compare.



Solidity Cheatsheet

The Solidity Cheatsheet is a valuable resource for all developers, from beginners to experts who just need a refresher.



Order of Precedence of Operators

The following is the order of precedence for operators, listed in order of evaluation.

Precedence	Description	Operator
1	Postfix increment and decrement	`++` , `--`
	New expression	`new <typename>`
	Array subscripting	`<array>[<index>]`
	Member access	`<object>.<member>`
	Function-like call	`<func>(<args...>)`
	Parentheses	`(<statement>)`
2	Prefix increment and decrement	`++` , `--`
	Unary minus	`-`
	Unary operations	`delete`
	Logical NOT	`!`
	Bitwise NOT	`~`
3	Exponentiation	`**`
4	Multiplication, division and modulo	`*` , `/` , `%`
5	Addition and subtraction	`+` , `-`
6	Bitwise shift operators	`<<` , `>>`
7	Bitwise AND	`&`
8	Bitwise XOR	`^`
9	Bitwise OR	`|`
10	Inequality operators	`<` , `>` , `<=` , `>=`
11	Equality operators	`==` , `!=`
12	Logical AND	`&&`
13	Logical OR	`||`
14	Ternary operator	`<conditional> ? <if-true> : <if-false>`
	Assignment operators	`=` , `|=` , `^=` , `&=` , `<<=` , `>>=` , `+=` , `-=` , `*=` , `/=` , `%=`
15	Comma operator	`,`

CertiK's Security Suite

As part of our mission to secure the Web3 world, CertiK provides a number of tools designed to help projects and investors take an end-to-end approach to security.

The CertiK Security Leaderboard helps Web3 users leverage the expertise of our auditing and security teams in order to equip themselves with a deeper knowledge of security risks. These users push the whole ecosystem to new heights, while we provide the data that helps them make informed decisions.

The recently updated Security Leaderboard 360 provides a comprehensive and up-to-the-minute picture of the security of thousands of onboarded projects. Thanks to a complete redesign, it's now easier than ever to access and interpret this data. We use quantitative methods to help retail investors make qualitative decisions about their investments. Check it out today at certik.com.

Skynet actively monitors hundreds of web3 platforms in real-time, using a combination of on - chain transaction monitoring and off-chain data such as social sentiment to ultimately arrive at a comprehensive security analysis. Skynet Premium – unveiled in 2021 – integrated continuously-evolving machine learning to grow in lockstep with the constantly shifting smart contract risk environment, getting more and more advanced as it encounters new threats and vulnerabilities. The Premium platform includes an analytics dashboard which enables enterprise clients to monitor and manage their risk in real-time.

CertiK KYC provides comprehensive and private identity verification for project teams. This process includes an ID authenticity inspection using AI-based detection systems, as well as liveness checks to ensure the individual is indeed real and matches the ID. CertiK will also undertake a live video call with each team member to verify their identity and other parameters as needed. As team anonymity increasingly enables high-risk behaviors, CertiK KYC helps to build accountability around projects to enable investors to move forward in trust. Projects that earn a Bronze, Silver, or Gold KYC Badge demonstrate to their community that they are willing to stand behind their project, sending a powerful message that they can be trusted to carry out the project's mission.

Penetration Testing is the final component of a comprehensive approach to securing crypto applications in a runtime environment. Our penetration testing services uncover even the smallest weaknesses by leveraging proprietary tooling, powered by an experienced team of ethical hackers.

CertiK Bug Bounty Program crowdsources intelligence from the world's top ethical hackers to uncover vulnerabilities before malicious actors can exploit them. CertiK's expert security engineers screen and qualify submissions and work with clients to implement the right fixes. Our 0% fee model reduces the payout pressure for projects and allows white hat hackers to receive the full bounty.

SkyTrace is an intelligent, intuitive tracing tool to help analyze and visualize transaction data across Ethereum and BSC wallets. This tool provides actionable insights into identifying and tracing suspicious flows to and from one's own personal wallet or a project's team wallet.

Supported Ecosystems at CertiK, we can audit all projects on all blockchains. A list of ecosystems we've worked with includes:

- Algorand
- Aptos
- Arbitrum
- Avalanche
- BNB Chain
- Cardano
- Cosmos
- Cronos
- Ethereum
- Fantom
- Ferrum
- Harmony
- IoTeX
- Near

SkyHarbor offers an all-in-one solution to secure and monitor your digital assets. With its advanced monitoring and threat detection system, SkyHarbor can quickly identify any vulnerabilities or suspicious activities in your system, ensuring that your assets are always safe and secure.

Get the most out of Web3 by partnering with **CertiK Advisory**. Our team of seasoned analysts deliver a comprehensive range of services, including technical evaluations, proprietary research, and strategy recommendations.