frontiers

# *Supplementary Material*

This collection of additional material contains the documentation of the systematic literature review in section 3. Regarding the extensive data extracted from the publications, we present the vulnerabilities organized in a consolidated taxonomy in section 1 and overviews of tools in section 2.

## Contents


| | | Page No. |
|---|---|---|
| 1 | Vulnerabilities in the Consolidated Taxonomy | 2 |
| 1.1 | Malicious Environment, Transactions or Input | 3 |
| 1.2 | Blockchain/Environment Dependency | 4 |
| 1.3 | Exception & Error Handling Disorders | 5 |
| 1.4 | Denial of Service | 6 |
| 1.5 | Resource Consumption & Gas Related Issues | 7 |
| 1.6 | Authentication & Access Control Vulnerabilities | 8 |
| 1.7 | Arithmetic Bugs | 9 |
| 1.8 | Bad Coding Quality and Programming Language Specifics | 10 |
| 1.9 | Environment Configuration Issues | 12 |
| 1.10 | Eliminated/Deprecated Vulnerabilities | 13 |
| 2 | Tools for the Analysis of Smart Contracts | 13 |
| 2.1 | Description of Tools | 14 |
| 2.2 | Open Source Tools | 38 |
| 2.3 | Properties and Methods of Tools | 43 |
| 3 | Quality Appraisal | 53 |
| 4 | References | 59 |

# 1 VULNERABILITIES IN THE CONSOLIDATED TAXONOMY

In this section, we detail the 54 vulnerabilities we extracted. The presentation follows our consolidated taxonomy that is structured into ten major classes.

| Code | Vulnerability |
|---|---|
| 1 | **Malicious Environment, Transactions or Input** |
| 1A | Reentrancy |
| 1B | Call to the unknown |
| 1C | Exact balance dependency |
| 1D | Improper data validation |
| 1E | Vulnerable DELEGATECALL |
| 2 | **Blockchain/Environment Dependency** |
| 2A | Timestamp dependency |
| 2B | Transaction-ordering dependency (TOD) |
| 2C | Bad random number generation |
| 2D | Leakage of confidential information |
| 2E | Unpredictable state (dynamic libraries) |
| 2F | Blockhash dependency |
| 3 | **Exception & Error Handling Disorders** |
| 3A | Unchecked low level call/send return values |
| 3B | Unexpected throw or revert |
| 3C | Mishandled out-of-gas exception |
| 3D | Assert, require or revert violation |
| 4 | **Denial of Service** |
| 4A | Frozen Ether |
| 4B | Ether lost in transfer |
| 4C | DoS with block gas limit reached |
| 4D | DoS by exception inside loop |
| 4E | Insufficient gas griefing |
| 5 | **Resource Consumption & Gas Issues** |
| 5A | Gas costly loops |
| 5B | Gas costly pattern |
| 5C | High gas consumption of variable data type or declaration |
| 5D | High gas consumption function type |
| 5E | Under-priced opcodes |

| Code | Vulnerability |
|---|---|
| 6 | **Authentication & Access Control Vulnerabilities** |
| 6A | Authorization via transaction origin |
| 6B | Unauthorized accessibility due to wrong function or state variable visibility |
| 6C | Unprotected self-destruction |
| 6D | Unauthorized Ether withdrawal |
| 6E | Signature based vulnerabilities |
| 7 | **Arithmetic Bugs** |
| 7A | Integer over- or underflow |
| 7B | Integer division |
| 7C | Integer bugs or arithmetic issues |
| 8 | **Bad Coding and Language Specifics** |
| 8A | Type cast |
| 8B | Coding error |
| 8C | Bad coding pattern |
| 8D | Deprecated source language features |
| 8E | Write to arbitrary storage location |
| 8F | Use of assembly |
| 8G | Incorrect inheritance order |
| 8H | Variable shadowing |
| 8I | Misleading source code |
| 8J | Missing logic, logical errors or dead code |
| 8K | Insecure contract upgrading |
| 8L | Inadequate or incorrect logging or documentation |
| 9 | **Environment Configuration Issues** |
| 9A | Short address |
| 9B | Outdated compiler version |
| 9C | Floating or no pragma |
| 9D | Token API violation |
| 9E | Ethereum update incompatibility |
| 9F | Configuration error |
| 10 | **Eliminated/Deprecated Vulnerabilities** |
| 10A | Callstack depth limit |
| 10B | Uninitialized storage pointer |
| 10C | Erroneous constructor name |

## 1.1 Malicious Environment, Transactions or Input (Class 1)

### 1A Reentrancy

*Alternate names:* Cross function race condition, race to empty, recursive call vulnerability

*Description:* The callee calls the caller back, before the initial call has completed and relevant checks and state changes have been performed. The classical example is an Ether transfer to a contract, whose fallback function is specifically crafted to call back the function emitting the original call.

*Discussed in 19 references:* NCC Group, 2018; SWC Registry, 2018; Atzei et al., 2017; Chen et al., 2020b,a; Demir et al., 2019; di Angelo and Salzer, 2019b; Dika and Nowostawski, 2018; Dingman et al., 2019b; Groce et al., 2020; Gupta et al., 2020a; Khan and Namin, 2020; Mense and Flatscher, 2018; Moona and Mathew, 2021; Praitheeshan et al., 2020b; Samreen and Alalfi, 2020b; Staderini et al., 2020; Tantikul and Ngamsuriyaroj, 2020; Zhang et al., 2020a

### 1B Call to the unknown

*Alternate names:* Unexpected fallback function

*Description:* An unrestricted CALL to an unknown address may lead to the execution of malicious code, e.g. the fallback function of the called contract. This vulnerability is a generalization of *1A Reentrancy*, where the callee executes a call back to the caller.

*Discussed in 10 references:* Atzei et al., 2017; Demir et al., 2019; Dika and Nowostawski, 2018; Dingman et al., 2019b; Gupta et al., 2020a; Mense and Flatscher, 2018; Moona and Mathew, 2021; Praitheeshan et al., 2020b; Samreen and Alalfi, 2020b; Staderini et al., 2020

### 1C Exact balance dependency

*Alternate names:* Forcing Ether to contracts, unexpected Ether, manipulated balance, pre-sent Ether

*Description:* If the logic of a contract depends on maintaining the exact balance in storage (instead of querying the environment for the current balance), attackers can manipulate the contract by increasing its balance without triggering its code. This can be achieved by using the contract's address as the beneficiary of a SELFDESTRUCT operation or as the receiver of a mining reward, or by predicting the contract's address and sending Ether before the contract gets deployed.

*Discussed in 7 references:* SWC Registry, 2018; Chen et al., 2020b,a; Dingman et al., 2019b; Gupta et al., 2020a; Tantikul and Ngamsuriyaroj, 2020; Zhang et al., 2020a

### 1D Improper data validation

*Alternate names:* untrustworthy data feeds, malicious libraries

*Description:* Failure to check data obtained from untrusted sources, such as external libraries or contracts, before using it.

*Discussed in 7 references:* Dika and Nowostawski, 2018; Groce et al., 2020; Gupta et al., 2020a; Mense and Flatscher, 2018; Praitheeshan et al., 2020b; Staderini et al., 2020; Tantikul and Ngamsuriyaroj, 2020

#### 1E Vulnerable DELEGATECALL

*Description:* Performing a DELEGATECALL or CALLCODE instruction to an untrusted address. Code invoked via these types of calls is run in the caller's context. A malicious callee can manipulate the caller's state variables and has full control over the balance.

*Discussed in 6 references:* SWC Registry, 2018; Chen et al., 2020a; Dingman et al., 2019b; Gupta et al., 2020a; Staderini et al., 2020; Zhang et al., 2020a

### 1.2 Blockchain/Environment Dependency (Class 2)

#### 2A Timestamp dependency

*Alternate names:* Time constraints, time manipulation

*Description:* The timestamp or the number of the block should not be used for precise calculations or critical time dependencies in the contract logic. Miners have control over these values and can manipulate them to a certain extent, or the values may be subject to changes for other reasons.

*Discussed in 17 references:* NCC Group, 2018; SWC Registry, 2018; Atzei et al., 2017; Chen et al., 2020a; Demir et al., 2019; di Angelo and Salzer, 2019b; Dika and Nowostawski, 2018; Dingman et al., 2019b; Groce et al., 2020; Gupta et al., 2020a; Khan and Namin, 2020; Mense and Flatscher, 2018; Moona and Mathew, 2021; Praitheeshan et al., 2020b; Staderini et al., 2020; Tantikul and Ngamsuriyaroj, 2020; Zhang et al., 2020a

#### 2B Transaction-ordering dependency (TOD)

*Alternate names:* Event-ordering bugs, front running, race condition

*Description:* The logic of a contract should not depend on the order, in which the transactions of a block are executed, as miners can compose the block in any order, and may do so deliberately for malicious reasons. Moreover, the state of a contract may change between the time a transaction is submitted to the network and the time it is included into a block, as other transactions, even when submitted later, may be prioritized by miners.

*Discussed in 16 references:* NCC Group, 2018; SWC Registry, 2018; Chen et al., 2020a; Demir et al., 2019; di Angelo and Salzer, 2019b; Dika and Nowostawski, 2018; Dingman et al., 2019b; Groce et al., 2020; Gupta et al., 2020a; Khan and Namin, 2020; Mense and Flatscher, 2018; Moona and Mathew, 2021; Praitheeshan et al., 2020b; Staderini et al., 2020; Tantikul and Ngamsuriyaroj, 2020; Zhang et al., 2020a

#### 2C Bad random number generation

*Alternate names:* Generating randomness, block number dependency, bad randomness

*Description:* Truly random numbers are important for applications like lotteries and games, but are difficult to generate. Using global blockchain variables, such as timestamps, as seed is open to manipulation by malicious miners who can influence global variables. Moreover, other transactions in the same block, or other contracts invoked in the same transaction have access to the same global variables and may guess the random number.

*Discussed in 13 references:* NCC Group, 2018; SWC Registry, 2018; Atzei et al., 2017; Chen et al., 2020b,a; Demir et al., 2019; di Angelo and Salzer, 2019b; Dika and Nowostawski, 2018; Dingman et al., 2019b; Groce et al., 2020; Gupta et al., 2020a; Mense and Flatscher, 2018; Staderini et al., 2020

#### 2D Leakage of confidential information

*Alternate names:* Lack of transaction privacy, unencrypted private data on-chain, private information leakage, nothing is secret, confidentiality failure, keeping secrets, data exposure, secrecy failure

*Description:* All transactions as well as the state of all contracts are visible to all miners and the world out side of the blockchain, even if the visibility of variables is set to private or internal in the source code. Therefore it is important to use encryption and commitment schemes to protect secret information.

*Discussed in 12 references:* SWC Registry, 2018; Atzei et al., 2017; Chen et al., 2020a; Dika and Nowostawski, 2018; Groce et al., 2020; Gupta et al., 2020a; Mense and Flatscher, 2018; Moona and Mathew, 2021; Samreen and Alalfi, 2020b; Staderini et al., 2020; Tantikul and Ngamsuriyaroj, 2020; Zhang et al., 2020a

#### 2E Unpredictable state (dynamic libraries)

*Description:* Calling contracts or libraries that can self-destruct or can be updated at a later stage will lead to unpredictable effects.

*Discussed in 5 references:* Atzei et al., 2017; di Angelo and Salzer, 2019b; Dingman et al., 2019b; Gupta et al., 2020a; Mense and Flatscher, 2018

#### 2F Blockhash dependency

*Alternate names:* Block number dependency

*Description:* Using the blockhash bears similar risks as using other block information like the timestamp or number and should not be used in critical operations, as miners can manipulate also the blockhash.

*Discussed in 4 references:* di Angelo and Salzer, 2019b; Dika and Nowostawski, 2018; Mense and Flatscher, 2018; Staderini et al., 2020

### 1.3 Exception & Error Handling Disorders (Class 3)

#### 3A Unchecked low level call/send return values

*Alternate names:* Unchecked/unsafe send, unchecked CALL return values, return value bypassing, send instead of transfer, unhandled exception

*Description:* Low level calls do not raise an exception on failure but return a Boolean status. Failing to check the return status and to handle errors adequately may lead to critical vulnerabilities.

*Discussed in 18 references:* NCC Group, 2018; SWC Registry, 2018; Atzei et al., 2017; Chen et al., 2020b,a; Demir et al., 2019; di Angelo and Salzer, 2019b; Dika and Nowostawski, 2018; Dingman et al., 2019b; Groce et al., 2020; Gupta et al., 2020a; Khan and Namin, 2020; Mense and Flatscher, 2018; Moona and Mathew, 2021; Praitheeshan et al., 2020b; Staderini et al., 2020; Tantikul and Ngamsuriyaroj, 2020; Zhang et al., 2020a

### 3B Unexpected throw or revert

*Alternate names:* DoS by external contract, unexpected-throw-DoS, DoS with failed call, DoS with unexpected revert/throw

*Description:* Calls to other contracts may fail and throw exceptions or revert the transaction. Not handling these situations adequately can lead to critical issues, such as DoS or lost funds.

*Discussed in 13 references:* NCC Group, 2018; SWC Registry, 2018; Atzei et al., 2017; Chen et al., 2020a; Demir et al., 2019; Dika and Nowostawski, 2018; Gupta et al., 2020a; Khan and Namin, 2020; Moona and Mathew, 2021; Samreen and Alalfi, 2020b; Staderini et al., 2020; Tantikul and Ngamsuriyaroj, 2020; Zhang et al., 2020a

### 3C Mishandled out-of-gas exception

*Alternate names:* Gasless send (subcategory of 3B)

*Description:* Calling another contract with too little gas will cause an out-of-gas exception, possibly resulting in a vulnerability if handled inadequately. A typical situation is the transfer of funds with Solidity's `send` method: The amount of 2300 gas supplied with the call will lead to an out-of-gas exception if the callee executes more than a few simple instructions.

*Discussed in 11 references:* Atzei et al., 2017; di Angelo and Salzer, 2019b; Dika and Nowostawski, 2018; Dingman et al., 2019b; Gupta et al., 2020a; Khan and Namin, 2020; Mense and Flatscher, 2018; Moona and Mathew, 2021; Samreen and Alalfi, 2020b; Staderini et al., 2020; Zhang et al., 2020a

### 3D Assert, require or revert violation

*Description:* Solidity's error handling via `assert`, `require` and `revert` (or its bytecode counterparts) needs to be used correctly to avoid abnormal function behavior.

*Discussed in 4 references:* SWC Registry, 2018; Gupta et al., 2020a; Staderini et al., 2020; Zhang et al., 2020a

## 1.4 Denial of Service (Class 4)

### 4A Frozen Ether

*Alternate names:* Greedy contract, locked Ether

*Description:* If a contract uses library code to provide access to its assets, then accidental or malicious self-destruction of the library contract may render the assets inaccessible.

*Discussed in 7 references:* Chen et al., 2020a; Dingman et al., 2019b; Khan and Namin, 2020; Praitheeshan et al., 2020b; Staderini et al., 2020; Tantikul and Ngamsuriyaroj, 2020; Zhang et al., 2020a

### 4B Ether lost in transfer

*Alternate names:* Transfer to orphan address

*Description:* Ether sent to an unused address becomes inaccessible as there is no matching private key.

*Discussed in 7 references:* Atzei et al., 2017; Chen et al., 2020a; di Angelo and Salzer, 2019b; Dingman et al., 2019b; Mense and Flatscher, 2018; Praitheeshan et al., 2020b; Staderini et al., 2020

#### 4C DoS with block gas limit reached

*Alternate names:* Block gas limit exceeded, DoS with unbounded operations, DoS by gaslimit

*Description:* The total gas spent by the transactions of a block is bounded by the block gas limit. Transactions exceeding this limit will not get processed.

*Discussed in 7 references:* NCC Group, 2018; SWC Registry, 2018; Chen et al., 2020a; Demir et al., 2019; Dingman et al., 2019b; Gupta et al., 2020a; Zhang et al., 2020a

#### 4D DoS by exception inside loop

*Alternate names:* DoS under external influence, wallet griefing causing out of gas exception

*Description:* Operations inside loops that may throw exceptions and depend on external transactions to succeed are open to DoS attacks by malicious users.

*Discussed in 2 references:* Chen et al., 2020b; Khan and Namin, 2020

#### 4E Insufficient gas griefing

*Description:* A contract relaying input from the caller to another contract can be manipulated by the caller, by providing just sufficient gas for the initial transaction but not for the sub-call.

*Discussed in 1 reference:* SWC Registry, 2018

### 1.5 Resource Consumption & Gas Related Issues (Class 5)

#### 5A Gas costly loops

*Alternate names:* Invariants in loops

*Description:* Unbounded and costly operations in loops should be avoided, to avoid inefficiency and potential DoS.

*Discussed in 6 references:* Chen et al., 2020b; Dingman et al., 2019b; Khan and Namin, 2020; Staderini et al., 2020; Tantikul and Ngamsuriyaroj, 2020; Zhang et al., 2020a

#### 5B Gas costly pattern

*Description:* Smart contract code should generally avoid inefficient operations and gas costly patterns that consume more gas than necessary. Several gas costly patterns are identified in Chen et al. (2017).

*Discussed in 4 references:* di Angelo and Salzer, 2019b; Khan and Namin, 2020; Mense and Flatscher, 2018; Praitheeshan et al., 2020b

#### 5C High gas consumption of variable data type or declaration

*Alternate names:* Byte array, invariant state variables not declared constant

*Description:* In Solidity, the data type `byte[]` consumes more gas than a byte array, due to padding rules. Invariants not declared `constant` consume more gas than necessary.

*Discussed in 3 references:* Chen et al., 2020b; Tantikul and Ngamsuriyaroj, 2020; Zhang et al., 2020a

#### 5D High gas consumption function type

*Alternate names:* Function declaration public instead of external

*Description:* In Solidity, functions not used in the contract but declared `public` instead of `external` require more gas on deployment than necessary.

*Discussed in 2 references:* Chen et al., 2020b; Zhang et al., 2020a

#### 5E Under-priced opcodes

*Description:* Contracts executing numerous under-prized opcodes may consume large amounts of computing resources, as the resource consumption is not reflected in the gas cost, which may lead to DoS attacks.

*Discussed in 2 references:* Chen et al., 2020a; Staderini et al., 2020

### 1.6 Authentication & Access Control Vulnerabilities (Class 6)

#### 6A Authorization via transaction origin

*Alternate names:* Authentication through `tx.origin`

*Description:* If a contract uses the external address initiating the transaction, `tx.origin`, for authentication purposes instead of the address of the immediate caller, contracts situated in the call chain between `tx.origin` and the vulnerable contract, gain access to the assets of the contract.

*Discussed in 15 references:* SWC Registry, 2018; Chen et al., 2020b,a; di Angelo and Salzer, 2019b; Dika and Nowostawski, 2018; Dingman et al., 2019b; Groce et al., 2020; Gupta et al., 2020a; Khan and Namin, 2020; Mense and Flatscher, 2018; Moona and Mathew, 2021; Praitheeshan et al., 2020b; Staderini et al., 2020; Tantikul and Ngamsuriyaroj, 2020; Zhang et al., 2020a

#### 6B Unauthorized accessibility due to wrong function or state variable visibility

*Alternate names:* No restricted write, implicit visibility level, non-public variables accessed by public/external functions, visibility of exposed functions, default/unintended/erroneous visibility

*Description:* In Solidity, functions inadvertently declared public/external will expose an entry point that may lead to unauthorized access by malicious users.

*Discussed in 9 references:* SWC Registry, 2018; Chen et al., 2020a; di Angelo and Salzer, 2019b; Dingman et al., 2019b; Groce et al., 2020; Gupta et al., 2020a; Praitheeshan et al., 2020b; Staderini et al., 2020; Zhang et al., 2020a

#### 6C Unprotected self-destruction

*Alternate names:* unexpected kill, destroyable contract

*Description:* Insufficient access control to a SELFDESTRUCT instruction may lead to the accidental or malicious destruction of a contract.

*Discussed in 9 references:* NCC Group, 2018; SWC Registry, 2018; Chen et al., 2020a; di Angelo and Salzer, 2019b; Gupta et al., 2020a; Khan and Namin, 2020; Praitheeshan et al., 2020b; Staderini et al., 2020; Zhang et al., 2020a

#### 6D Unauthorized Ether withdrawal

*Description:* Insufficient access control to the withdrawal functionality of a contract can lead to the theft of assets.

*Discussed in 8 references:* SWC Registry, 2018; Chen et al., 2020a; Gupta et al., 2020a; Khan and Namin, 2020; Praitheeshan et al., 2020b; Staderini et al., 2020; Tantikul and Ngamsuriyaroj, 2020; Zhang et al., 2020a

#### 6E Signature based vulnerabilities

*Alternate names:* Insufficient signature information

*Description:* Includes issues like a missing protection against signature replay attacks, lack of proper signature verification, hash collisions with multiple variable length arguments, signature malleability, and signatures with a wrong parameter.

*Discussed in 5 references:* SWC Registry, 2018; Chen et al., 2020a; Gupta et al., 2020a; Staderini et al., 2020; Zhang et al., 2020a

### 1.7 Arithmetic Bugs (Class 7)

#### 7A Integer over- or underflow

*Description:* Integer over- and underflows occur when arithmetic operations exceed the maximum or minimum of integer types. Unless this exception is caught, a wrap-around occurs.

*Discussed in 10 references:* SWC Registry, 2018; Chen et al., 2020a; Dingman et al., 2019b; Gupta et al., 2020a; Khan and Namin, 2020; Praitheeshan et al., 2020b; Samreen and Alalfi, 2020b; Staderini et al., 2020; Tantikul and Ngamsuriyaroj, 2020; Zhang et al., 2020a

#### 7B Integer division

*Alternate names:* Floating point & precision

*Description:* Handling floating point numbers, represented as integers, incorrectly may lead to inaccuracies, errors and vulnerabilities. The same holds for the result of integer divisions, which are rounded down in Solidity.

*Discussed in 5 references:* Dingman et al., 2019b; Groce et al., 2020; Gupta et al., 2020a; Tantikul and Ngamsuriyaroj, 2020; Zhang et al., 2020a

### 7C Integer bugs or arithmetic issues

*Alternate names:* Unchecked math, integer sign, integer truncation, wrong operator

*Description:* The conversion between signed and unsigned integers or between integers of different length may lead to incorrect results, as can the incorrect usage of arithmetic operators.

*Discussed in 5 references:* NCC Group, 2018; di Angelo and Salzer, 2019b; Khan and Namin, 2020; Mense and Flatscher, 2018; Zhang et al., 2020a

## 1.8 Bad Coding Quality and Programming Language Specifics (Class 8)

### 8A Type cast

*Alternate names:* Unsafe type inference

*Description:* To interact with another contract, the programmer specifies its interface and uses it to typecast addresses. This allows the Solidity compiler to construct appropriate low level calls. However, the compiler does not (cannot) check whether the interface actually matches the other contract, even though the programmer might have the false impression that the contract is correctly typed. Calling a contract with a wrong interface specification may result in the invocation of unintended entry points, with arguments interpreted differently than expected.

*Discussed in 8 references:* Atzei et al., 2017; Chen et al., 2020b,a; Dingman et al., 2019b; Gupta et al., 2020a; Mense and Flatscher, 2018; Samreen and Alalfi, 2020b; Staderini et al., 2020

### 8B Coding error

*Alternate names:* Immutable bugs, typographical error, coding bug

*Description:* Coding errors, including typos and other developer errors.

*Discussed in 8 references:* SWC Registry, 2018; Atzei et al., 2017; Dika and Nowostawski, 2018; Dingman et al., 2019b; Groce et al., 2020; Gupta et al., 2020a; Mense and Flatscher, 2018; Zhang et al., 2020a

### 8C Bad coding pattern

*Alternate names:* Dynamic array elements, continue-statements in do-while-statements, nonstandard naming conventions

*Description:* Use of various coding patterns that are regarded bad style.

*Discussed in 5 references:* di Angelo and Salzer, 2019b; Dika and Nowostawski, 2018; Groce et al., 2020; Mense and Flatscher, 2018; Zhang et al., 2020a

### 8D Deprecated source language features

*Description:* The use of deprecated language features is discouraged, as it may lead to known errors and vulnerabilities that were the cause for deprecation.

*Discussed in 4 references:* SWC Registry, 2018; di Angelo and Salzer, 2019b; Gupta et al., 2020a; Zhang et al., 2020a

### 8E Write to arbitrary storage location

*Description:* Due to quirks of the programming language (most notably Solidity) and programming mistakes, an attacker is able to modify arbitrary storage locations, like an owner variable used for access control. As an example, Solidity allocates complex data types like structs, mappings and arrays statically in storage, even when declaring a local variable of such a type in a function. Using the variable before initializing it will give access to the first storage cell.

*Discussed in 4 references:* SWC Registry, 2018; Chen et al., 2020b; Gupta et al., 2020a; Zhang et al., 2020a

### 8F Use of assembly

*Alternate names:* Arbitrary jump with function type variable, specify function variable as any type, returning results using assembly code in constructor

*Description:* The use of assembly instructions in high level code is discouraged, as it can lead to various critical vulnerabilities.

*Discussed in 3 references:* SWC Registry, 2018; Gupta et al., 2020a; Zhang et al., 2020a

### 8G Incorrect inheritance order

*Description:* Solidity supports multiple inheritance, which can lead to an ambiguity called *Diamond Problem*, if two or more base contracts define the same function. This is resolved using *C3 Linearization*, which leads to a deterministic method resolution order. Programmers not aware of this aspect may base their code on wrong assumptions, leading to unexpected behavior and vulnerabilities.

*Discussed in 3 references:* SWC Registry, 2018; Gupta et al., 2020a; Zhang et al., 2020a

### 8H Variable shadowing

*Alternate names:* Hidden state variables

*Description:* Solidity allows for ambiguous naming of variables when inheritance is used, which can lead to errors and vulnerablities that are difficult to identify in complex contract systems.

*Discussed in 3 references:* SWC Registry, 2018; Gupta et al., 2020a; Zhang et al., 2020a

### 8I Misleading source code

*Alternate names:* Right-to-left-override control character (U+202E), API inconsistency, hidden built-in symbols

*Description:* Malicious actors may apply various strategies to confuse or hide malicious contract code behavior and trick users.

*Discussed in 3 references:* SWC Registry, 2018; Groce et al., 2020; Zhang et al., 2020a

### 8J Missing logic, logical errors or dead code

*Alternate names:* Irrelevant code, code with no effects, undefined behavior, presence of unused variables

*Description:* Missing logic or logical errors or dead code can confuse or lead to unwanted behavior and vulnerabilities

*Discussed in 2 references:* SWC Registry, 2018; Groce et al., 2020

#### 8K Insecure contract upgrading

*Alternate names:* Upgradable contract, patching

*Description:* Even though making contracts upgradable, allows for fixing existing vulnerabilities, it counters the goal of completely decentralized smart contracts. If a contract developer becomes malicious or is compromized, the updated contract can become malicious. The updating methods themselves are also hard to implement correctly and without flaws.

*Discussed in 2 references:* Chen et al., 2020a; Groce et al., 2020

#### 8L Inadequate or incorrect logging or documentation

*Description:* Inadequate logging and documentation can confuse, lead to logical errors or unexpected behavior and hampers auditing or testing the code for vulnerabilities.

*Discussed in 1 reference:* Groce et al., 2020

### 1.9 Environment Configuration Issues (Class 9)

#### 9A Short address

*Description:* The EVM pads with zeroes if the provided address is shorter than the required length. Certain addresses with trailing zeroes are vulnerable to attackers, if client sanitation checks are inadequate.

*Discussed in 5 references:* NCC Group, 2018; Chen et al., 2020a; Gupta et al., 2020a; Staderini et al., 2020; Zhang et al., 2020a

#### 9B Outdated compiler version

*Description:* Using outdated compiler versions is strongly discouraged, as old compiler bugs or updates can lead to vulnerabilities in compiled smart contract code.

*Discussed in 5 references:* SWC Registry, 2018; Chen et al., 2020a; Dingman et al., 2019b; Gupta et al., 2020a; Tantikul and Ngamsuriyaroj, 2020

#### 9C Floating or no pragma

*Description:* The same compiler version and flags, that were originally used for tests should be used when deploying the smart contract. Therefore the pragma should be locked to the correct compiler version.

*Discussed in 3 references:* SWC Registry, 2018; Gupta et al., 2020a; Zhang et al., 2020a

#### 9D Token API violation

*Description:* Token contracts should meet applicable token standards such as ERC20 or ERC721 to avoid vulnerabilities and problems interacting with other contracts.

*Discussed in 2 references:* Tantikul and Ngamsuriyaroj, 2020; Zhang et al., 2020a

#### 9E Ethereum update incompatibility

*Alternate names:* Message call with hardcoded gas amount

*Description:* Gas cost for opcodes might change significantly in future Ethereum hard forks and might break already deployed contract systems with fixed gas cost assumptions.

*Discussed in 2 references:* SWC Registry, 2018; Gupta et al., 2020a

#### 9F Configuration error

*Description:* Bad configuration of the smart contract application tool chain can lead to errors and vulnerabilities, even if the smart contract itself is correct.

*Discussed in 1 reference:* Groce et al., 2020

### 1.10 Eliminated/Deprecated Vulnerabilities (Class 10)

#### 10A Callstack depth limit

*Alternate names:* Stack size limit

*Description:* Eliminated by Ethereum upgrade (Oct 18, 2016)

*Discussed in 9 references:* Atzei et al., 2017; Chen et al., 2020a; di Angelo and Salzer, 2019b; Dika and Nowostawski, 2018; Dingman et al., 2019b; Gupta et al., 2020a; Khan and Namin, 2020; Praitheeshan et al., 2020b; Tantikul and Ngamsuriyaroj, 2020

#### 10B Uninitialized storage pointer

*Alternate names:* Uninitialized local/state variables

*Description:* Eliminated by Solidity compiler, starting version 0.5.0

*Discussed in 5 references:* SWC Registry, 2018; Chen et al., 2020a; Dingman et al., 2019b; Gupta et al., 2020a; Zhang et al., 2020a

#### 10C Erroneous constructor name

*Description:* Eliminated in Solidity version 0.4.22 by introducing the new keyword `constructor`

*Discussed in 3 references:* SWC Registry, 2018; Chen et al., 2020a; Zhang et al., 2020a

## 2 TOOLS FOR THE ANALYSIS OF SMART CONTRACTS

In section 2.1, we present the 140 tools we identified by providing name, date, description, and references. Additional information for open-source tools can be found in section 2.2. Finally, in section 2.3 we indicate for all tools the respective properties and methods used to analyze smart contracts.

## 2.1 Description of Tools

### ABBE (2019-11)

Behavior based analysis for vulnerability detection in Solidity smart contracts, based on AST generation and transaction analysis

*Primary study introducing the tool:* Nguyen et al., 2019

### ADF-GA (2020-06)

*All-uses Data Flow criterion based test case generation using Genetic Algorithm (ADF-GA)* framework for Solidity smart contracts, based on CFG construction and analysis, data flow analysis and program instrumentation

*Primary study introducing the tool:* Zhang et al., 2020b

### ÆGIS (2020-08, open source)

Dynamic analysis to detect and protect smart contracts against attacks at runtime

*Primary study introducing the tool:* Ferreira Torres et al., 2019, 2020

### Annotary (2019-09)

Concolic execution framework for vulnerability detection in EVM bytecode with Solidity source code annotations

*Primary study introducing the tool:* Weiss and Schütte, 2019

*Discussed in 2 surveys:* Kim and Ryu, 2020; Tolmach et al., 2020

### CESC (2019-05)

Automatic detection of specific smart contract vulnerabilities, that are open to concurrency exploitation from malicious mining attacks

*Primary study introducing the tool:* Li, 2019

### Clairvoyance (2020-06)

Reentrancy vulnerability detection in Solidity smart contracts, based on cross-function and cross-contract static analysis, including cross-contract call graph and CFG (XCFG) generation, light-weight symbolic execution and static taint analysis

*Primary study introducing the tool:* Xue et al., 2020; Ye et al., 2020

### *Comparable vector encoding and matching* (2021-01)

Static smart contract bytecode analysis, based on graph embedding for quantitatively comparable vector encoding and vulnerability detection by measuring and matching similarities to known vulnerable vectors

*Primary study introducing the tool:* Huang et al., 2021

### ConCert (2020-01, open source)

Smart contract verification framework in Coq for functional smart contract languages

*Primary study introducing the tool:* Annenkov et al., 2020

*Discussed in 1 survey:* Tolmach et al., 2020

### Conkas (2021-03, open source)

Modular static analysis tool for EVM bytecode or Solidity symbolic execution based on Z3 as the SMT Solver and Rattle as intermediate representation

*Discussed in 1 survey:* Ferreira et al., 2020a

### *(Dr. Y's Ethereum) Contract Analyzer* (2016-09, open source)

Early proof of concept for static EVM bytecode analysis based on symbolic execution

*Discussed in 2 surveys:* di Angelo and Salzer, 2019b; Miller et al., 2018

### ContractFuzzer (2018-08, open source)

EVM smart contract fuzzer for vulnerability detection, based on EVM instrumentation and runtime behavior analysis

*Primary study introducing the tool:* Jiang et al., 2018; Mei et al., 2019

*Discussed in 9 surveys:* Hu et al., 2021a; Kim and Ryu, 2020; Vacca et al., 2020; Almakhour et al., 2020b; Chen et al., 2020a; Khan and Namin, 2020; Perez and Livshits, 2019; Samreen and Alalfi, 2020b; Xu et al., 2020

### ContractGuard (GuardStrike) (2018-06)

Solidity smart contract vulnerability detection based on fuzz testing, symbolic execution, CFG generation and analysis and SMT solving.
Live deployment: `https://contract.guardstrike.com/#/scan`

*Primary study introducing the tool:* ContractGuard, 2018

### ContractGuard (IDS) (2019-10)

Embedded intrusion detection system (IDS) to defend Ethereum smart contracts against attacks, based on the detection of abnormal control flows.
Experiments: `https://github.com/contractguard/experiments`

*Primary study introducing the tool:* Wang et al., 2019c

### ContractLarva (2019-08, open source)

Solidity runtime verification based on dynamic event automata specification

*Primary study introducing the tool:* Ellul and Pace, 2018

*Discussed in 10 surveys:* Kim and Ryu, 2020; Singh et al., 2020; Tolmach et al., 2020; Vacca et al., 2020; Almakhour et al., 2020b,a; Chen et al., 2020a; di Angelo and Salzer, 2019b; Durieux et al., 2020; López Vivar et al., 2020

### ContractMut (2020-03, open source)

Solidity contract mutation generation and gas consumption evaluation

*Primary study introducing the tool:* Hartel and Schumi, 2020

*Discussed in 1 survey:* Kim and Ryu, 2020

### ContractVis (2019-07)

Replay script generator and execution engine for analysis of historic transactions of smart contracts

*Primary study introducing the tool:* Hartel and van Staalduinen, 2019

*Discussed in 1 survey:* Kim and Ryu, 2020

### ContractWard (2020-01)

Smart contract vulnerability detection based on machine learning techniques, namely the supervised ensemble classification algorithm XGBoost trained on balanced training sets with the sampling method SMOTETomek

*Primary study introducing the tool:* Wang et al., 2020b

### ContraMaster/Vultron (2019-05, open source)

Solidity smart contract fuzzing framework for vulnerability detection and exploit generation

*Primary study introducing the tool:* Wang et al., 2019a, 2020a

*Discussed in 2 surveys:* Kim and Ryu, 2020; Khan and Namin, 2020

### *Convolutional neural network* (2018-07)

Smart contract analysis by machine learning based on RGB color code representation and Ethereum compiler bug detection

*Primary study introducing the tool:* Huang, 2018

*Discussed in 1 survey:* Kim and Ryu, 2020

### DappGuard (2017-05)

Student project. The report describes a system for monitoring and protecting smart contracts on the Ethereum blockchain. Has not been implemented, the repository contains just a few scripts for checking the feasibility of the idea. Nevertheless reviewed in several papers as if it really existed, based on the report.

*Primary study introducing the tool:* Cook et al., 2017

*Discussed in 5 surveys:* Kim and Ryu, 2020; Chen et al., 2020a; di Angelo and Salzer, 2019b; Huang et al., 2019; Khan and Namin, 2020

### *Deductive verification with Why3* (2019-10)

Deductive verification on smart contracts, based on Why3 and WhyML intermediate representation

*Primary study introducing the tool:* Nehaï and Bobot, 2019

*Discussed in 2 surveys:* Hu et al., 2021a; Tolmach et al., 2020

### DefectChecker (2021-01, open source)

Symbolic execution-based tool for contract defect detection on EVM bytecode

*Primary study introducing the tool:* Chen et al., 2021

### Deviant (2019-07)

Mutation generator and testing tool for Solidity smart contracts

*Primary study introducing the tool:* Chapman et al., 2019

### E-EVM (2018-01, open source)

EVM Emulator

*Primary study introducing the tool:* Norvill et al., 2018

*Discussed in 2 surveys:* di Angelo and Salzer, 2019b; Durieux et al., 2020

### EASYFLOW (2018-05, open source)

Arithmetic overflow detector for EVM bytecode based on taint analysis

*Primary study introducing the tool:* Gao et al., 2019b

*Discussed in 2 surveys:* Kim and Ryu, 2020; Khan and Namin, 2020

### ECFChecker (2017-10, open source)

Dynamic monitor for effectively callback free (ECF) Ethereum smart contract verification

*Primary study introducing the tool:* Grossman et al., 2017

*Discussed in 6 surveys:* Hu et al., 2021a; Kim and Ryu, 2020; Tolmach et al., 2020; Chen et al., 2020a; Huang et al., 2019; Miller et al., 2018

### Echidna (2020-01, open source)

Extendable Ethereum smart contract fuzzer and tester for violations in assertions and custom properties

*Primary study introducing the tool:* Grieco et al., 2020

*Discussed in 4 surveys:* Hu et al., 2021a; Durieux et al., 2020; Sayeed et al., 2020; Ye et al., 2019a

### Echidna-Parade (2021-07, open source)

Multicore fuzzing implementation of Echidna fuzzer including configuration diversification to improve performance and effectiveness

*Primary study introducing the tool:* Groce and Grieco, 2021

### Erays (2018-10, open source)

Ethereum smart contract reverse engineering tool producing high level pseudocode to support manual analysis

*Primary study introducing the tool:* Zhou et al., 2018b

*Discussed in 5 surveys:* Chen et al., 2020a; di Angelo and Salzer, 2019b; Durieux et al., 2020; Huang et al., 2019; López Vivar et al., 2020

### Ethainter (2020-04, open source)

Dynamic blockchain smart contract transaction analyzer allowing for composite attack detection and automated exploit generation based on data flow and taint analysis

*Primary study introducing the tool:* Brent et al., 2020

*Discussed in 1 survey:* Tolmach et al., 2020

### ETHBMC (2020-04, open source)

Bounded model checker for EVM bytecode analysis and automatic vulnerability scanning based on symbolic execution

*Primary study introducing the tool:* Frank et al., 2020

*Discussed in 1 survey:* Tolmach et al., 2020

### Ether* (S-GRAM) (2018-09)

Prototype based on S-gram, a combination of N- gram language modeling and static semantic metadata generation for statistical analysis and auditing of Ethereum smart contracts

*Primary study introducing the tool:* Liu et al., 2018b

*Discussed in 3 surveys:* Kim and Ryu, 2020; di Angelo and Salzer, 2019b; Durieux et al., 2020

### Etherolic (2020-03)

Dynamic Ethereum smart contract security analysis approach for vulnerability identification, exploitation and protection method recognition, based on taint tracking and concolic testing

*Primary study introducing the tool:* Ashouri, 2020

### Ethersplay (2018-05, open source)

Disassembler, graphical control flow graph generation and related analysis tools to support manual analysis of Ethereum smart contracts in bytecode

*Discussed in 1 survey:* Durieux et al., 2020

### EtherTrust (2018-05, open source)

Static analyzer and verifier for EVM bytecode with soundness guarantee for certain security properties, such as single entrancy and independence of the transaction environment

*Primary study introducing the tool:* Grishchenko et al., 2018b,a

*Discussed in 8 surveys:* Hu et al., 2021a; Kim and Ryu, 2020; Almakhour et al., 2020b; Chen et al., 2020a; di Angelo and Salzer, 2019b; Durieux et al., 2020; Khan and Namin, 2020; López Vivar et al., 2020

### EthIR (2020-05, open source)

Framework for Ethereum bytecode analysis based on extension of Oyente and control flow graphs (CFG) to generate rule-based representation (RBR)

*Primary study introducing the tool:* Albert et al., 2018

*Discussed in 8 surveys:* Hu et al., 2021a; Almakhour et al., 2020b; Chen et al., 2020a; di Angelo and Salzer, 2019b; Durieux et al., 2020; López Vivar et al., 2020; Praitheeshan et al., 2020b; Samreen and Alalfi, 2020b

### eThor (2020-10, open source)

Sound static analysis and verification for Ethereum smart contracts based on HoRSt, a modular, high-level analysis specification language framework allowing for abstract semantics in the form of Horn clauses

*Primary study introducing the tool:* Schneidewind et al., 2020a,b

*Discussed in 2 surveys:* Hu et al., 2021a; Tolmach et al., 2020

### ETHPLOIT (2020-02)

Ethereum smart contract fuzzer for automatic vulnerability detection and exploit generation

*Primary study introducing the tool:* Zhang et al., 2020c

*Discussed in 1 survey:* Samreen and Alalfi, 2020b

### ETHRACER (2018-08, open source)

Automatic Ethereum bytecode analysis tool for the identification of event-ordering bugs based on a combination of randomized fuzzing of event sequences and dynamic symbolic execution optimisation

*Primary study introducing the tool:* Kolluri et al., 2019

*Discussed in 2 surveys:* Hu et al., 2021a; Kim and Ryu, 2020

### EthScope (2020-10)

Scalable Ethereum attack detection framework, allowing for large-scale analysis of Ethereum smart contract transactions, based on three components, namely data aggregation of blockchain information, a transaction replay engine and a code instrumentation framework
`https://hub.docker.com/r/swaywu/ethscope`

*Primary study introducing the tool:* Wu et al., 2020

### *EVM "Memory" Modeling* (2020-09, open source)

Precise modeling of EVM memory, recovering high-level structures, such as arrays, buffers, call arguments via deep modeling of the flow of values enabling vulnerability detection and gas cost computation

*Primary study introducing the tool:* Lagouvardos et al., 2020

### EVM* (2019-02)

EVM instrumentation for monitoring and prevention of smart contract exploitation

*Primary study introducing the tool:* Ma et al., 2019

*Discussed in 2 surveys:* Singh et al., 2020; Tolmach et al., 2020

### *(Grishchenko et al.'s) F* EVM small-step semantic* (2018-04, open source)

Small-step semantics of EVM bytecode, formalized in the F* proof assistant, a functional programming language aimed at program verification

*Primary study introducing the tool:* Grishchenko et al., 2018c

*Discussed in 5 surveys:* Hu et al., 2021a; Tolmach et al., 2020; Chen et al., 2020a; Huang et al., 2019; Miller et al., 2018

### *(Bhargavan et al.'s)F* Verification* (2016-10)

Framework for security analysis and verification of Ethereum smart contracts by translation to F*

*Primary study introducing the tool:* Bhargavan et al., 2016

*Discussed in 11 surveys:* Hu et al., 2021a; Kim and Ryu, 2020; Tolmach et al., 2020; Almakhour et al., 2020b,a; Dika and Nowostawski, 2018; Garfatta et al., 2021; Huang et al., 2019; Khan and Namin, 2020; Moona and Mathew, 2021; Praitheeshan et al., 2020b

### FAIRCON (2020-05, open source)

Semi-automated verification of smart contract fairness properties including truthfulness, efficiency, optimality, and collusion-freeness, based on code instrumentation, intermediate modeling language translation, symbolic execution and SMT solving

*Primary study introducing the tool:* Liu et al., 2020b

### FSFC (2020-03)

Unified input Filter-based Secure Framework for Ethereum smart contracts, based on dynamic identification and prevention of bad input processing by deployment of input filters for live contracts

*Primary study introducing the tool:* Wang et al., 2020d

### FSMC (2019-05, open source)

FSM synthesis for automatic correct-by-construction Solidity smart contract generation

*Primary study introducing the tool:* Suvorov and Ulyantsev, 2019

*Discussed in 1 survey:* Kim and Ryu, 2020

### FSolidM/VeriSolid Framework (2017-10, open source)

Framework with a graphical user interface to specify and design smart contracts as finite state machines (FSMs)and automatically generate the corresponding Solidity code, including the support for a set of security and functionality plugins

*Primary study introducing the tool:* Mavridou and Laszka, 2018a,b; Mavridou et al., 2019; Nelaturu et al., 2020

*Discussed in 12 surveys:* Hu et al., 2021a; Kim and Ryu, 2020; Singh et al., 2020; Tolmach et al., 2020; Vacca et al., 2020; Almakhour et al., 2020b; Chen et al., 2020a; di Angelo and Salzer, 2019b; Durieux et al., 2020; Garfatta et al., 2021; Huang et al., 2019; Khan and Namin, 2020

### FSPVM-E/FEther (2018-07)

Hybrid formal security verification system implemented in Coq for Ethereum-based Solidity smart contracts, combining symbolic execution and higher-order logic theorem proving. The execution engine FEther supports static, concolic, and selective symbolic execution simultaneously

*Primary study introducing the tool:* Yang and Lei, 2018a,b, 2019b,a; Yang et al., 2020b

*Discussed in 6 surveys:* Hu et al., 2021a; Kim and Ryu, 2020; Tolmach et al., 2020; Almakhour et al., 2020b,a; Praitheeshan et al., 2020b

### GasChecker (2020-03)

Identification of ten gas-inefficient smart contract bytecode patterns on a scalable platform by parallelizing symbolic execution with MapReduce and load balancing strategies

*Primary study introducing the tool:* Chen et al., 2020d

*Discussed in 2 surveys:* Hu et al., 2021a; Vacca et al., 2020

### GASOL (2020-02, open source)

Enhancement of GASTAP's smart contract gas analysis, based on cost models of EVM or Solidity instructions to infer upper gas bounds, identify under-optimized storage patterns and provide automatic function optimization

*Primary study introducing the tool:* Albert et al., 2020a

*Discussed in 1 survey:* Hu et al., 2021a

### Gasper (2017-02)

Detection of gas-costly patterns, such as dead code, opaque predicates and expensive loop operations in smart contract bytecode, based on symbolic execution

*Primary study introducing the tool:* Chen et al., 2017

*Discussed in 11 surveys:* Hu et al., 2021a; Kim and Ryu, 2020; Vacca et al., 2020; Almakhour et al., 2020b; di Angelo and Salzer, 2019b; Dika and Nowostawski, 2018; Durieux et al., 2020; Garfatta et al., 2021; Khan and Namin, 2020; Moona and Mathew, 2021; Praitheeshan et al., 2020b

### GasReducer (2018-05)

Enhancement of Gasper to identify 24 gas-costly patterns in an emulation-based framework for smart contract bytecode

*Primary study introducing the tool:* Chen et al., 2018a

*Discussed in 2 surveys:* Hu et al., 2021a; Kim and Ryu, 2020

### GASTAP (2019-10)

Gas Analysis for smart contract bytecode or Solidity to infer upper gas bounds for all public functions to identify potential out-of-gas exceptions.
Live deployment: `https://costa.fdi.ucm.es/gastap`

*Primary study introducing the tool:* Albert et al., 2019b

*Discussed in 2 surveys:* Hu et al., 2021a; Kim and Ryu, 2020

### Gazfuzzer (2020-05)

Gas consumption driven fuzzing for security vulnerability detection of smart contracts, based on gas allowance manipulation and mutation to expose gas-oriented exception security vulnerabilities

*Primary study introducing the tool:* Ashraf et al., 2020

### Gigahorse (2019-01, open source)

Toolchain containing smart contract bytecode decompiler to higher-level function-based 3-address code representation, similar to LLVM IR, for data- and control-flow analysis.
Live deployment: `https://contract-library.com/`

*Primary study introducing the tool:* Grech et al., 2019

*Discussed in 1 survey:* Hu et al., 2021a

### GNNSCVulDetector (2020-06, open source)

Solidity smart contract vulnerability detection (reentrancy, timestamp dependence, and infinite loop), based on a degree-free graph convolutional neural network (DR-GCN) and temporal message propagation network (TMP) to learn from the normalized graphs

*Primary study introducing the tool:* Zhuang et al., 2020

### HARVEY/BRAN (2018-08)

Industrial greybox fuzzer extension based on input prediction and on-demand transaction sequence fuzzing for vulnerability detection and path discovery. HARVEY has been subsequently integrated with BRAN, an open source static abstract-interpretation framework

*Primary study introducing the tool:* Wüstholz and Christakis, 2020a,b

*Discussed in 1 survey:* Kim and Ryu, 2020

### HONEYBADGER (2019-02, open source)

Honeypot detection based on classified honeypot techniques and symbolic execution

*Primary study introducing the tool:* Torres et al., 2019

*Discussed in 4 surveys:* Kim and Ryu, 2020; Tolmach et al., 2020; Ferreira et al., 2020a; Durieux et al., 2020

### Hydra (2017-11, open source)

Automated bug bounty incentive framework based on a variant of classical N-version (redundant) programming for modeling and detection of security-critical Solidity smart contract vulnerabilities

*Primary study introducing the tool:* Breidenbach et al., 2018, 2019

*Discussed in 3 surveys:* Kim and Ryu, 2020; Singh et al., 2020; Huang et al., 2019

### ILF (2019-11, open source)

Imitation Learning based Fuzzer (ILF) for smart contracts, with the aim to provide effective and fast coverage-based fuzzing based on neural network machine learning with symbolic execution integration

*Primary study introducing the tool:* He et al., 2019

*Discussed in 1 survey:* Hu et al., 2021a

### *Isabelle/HOL based framework* (2017-03, open source)

Frameworks using the interactive theorem prover Isabelle/HOL by formally defining the EVM in a compatible language, such as Lem, to enable smart contract verification

*Primary study introducing the tool:* Hirai, 2017; Amani et al., 2018

*Discussed in 11 surveys:* Hu et al., 2021a; Kim and Ryu, 2020; Tolmach et al., 2020; Almakhour et al., 2020b,a; Chen et al., 2020a; Garfatta et al., 2021; Huang et al., 2019; Khan and Namin, 2020; Miller et al., 2018; Praitheeshan et al., 2020b

### JAVADITY (2018-07, open source)

Verifying smart contracts by translating Solidity into Java and using the deductive Java verification tool KeY

*Primary study introducing the tool:* Ahrendt et al., 2019

*Discussed in 1 survey:* Tolmach et al., 2020

### KEVM (2019-07, open source)

Formal semantic specification model of the EVM using the K Framework to support further formal analysis and verification including backends for symbolic execution

*Primary study introducing the tool:* Hildenbrandt et al., 2018

*Discussed in 9 surveys:* Hu et al., 2021a; Vacca et al., 2020; Chen et al., 2020a; di Angelo and Salzer, 2019b; Durieux et al., 2020; Huang et al., 2019; Khan and Namin, 2020; Miller et al., 2018; Praitheeshan et al., 2020b

### *KEVM Verifier* (2018-10, open source)

Smart contract bytecode verification tool based on KEVM

*Primary study introducing the tool:* Chen et al., 2018b; Park et al., 2018

*Discussed in 4 surveys:* Kim and Ryu, 2020; Singh et al., 2020; Tolmach et al., 2020; Chen et al., 2020a

### KSolidity (2019-07, open source)

Executable operational semantics of Solidity in the K Framework providing a formal specification of smart contracts to allow for the definition of semantic-level security properties for formal verification of Solidity smart contracts

*Primary study introducing the tool:* Jiao et al., 2020

*Discussed in 1 survey:* Tolmach et al., 2020

### *LSTM Machine learning* (2018-11, open source)

Sequential learning of smart contract issues and large scale analysis using the machine learning approach long-short term memory (LSTM)

*Primary study introducing the tool:* Tann et al., 2018

*Discussed in 2 surveys:* Kim and Ryu, 2020; Chen et al., 2020a

### MadMax (2018-09, open source)

Gas-focused static vulnerability analysis in Ethereum smart contract bytecode, based on the Gigahorse toolchain, applying decompilation and declarative approaches for higher-level analysis

*Primary study introducing the tool:* Grech et al., 2018

*Discussed in 10 surveys:* Hu et al., 2021a; Kim and Ryu, 2020; Tolmach et al., 2020; Vacca et al., 2020; Almakhour et al., 2020b; Chen et al., 2020a; Durieux et al., 2020; Khan and Namin, 2020; López Vivar et al., 2020; Perez and Livshits, 2019

### MAIAN (2018-03, open source)

Bytecode analysis for identification of three types of vulnerable smart contracts, i.e. prodigal, suicidal and greedy, based on reasoning about transaction trace properties, employing symbolic analysis

*Primary study introducing the tool:* Nikolić et al., 2018

*Discussed in 21 surveys:* Hu et al., 2021a; Kim and Ryu, 2020; Singh et al., 2020; Tolmach et al., 2020; Vacca et al., 2020; Ferreira et al., 2020a; Almakhour et al., 2020b; Chen et al., 2020a; di Angelo and Salzer, 2019b; Durieux et al., 2020; Garfatta et al., 2021; Huang et al., 2019; Khan and Namin, 2020; López Vivar et al., 2020; Miller et al., 2018; Moona and Mathew, 2021; Perez and Livshits, 2019; Praitheeshan et al., 2020b; Samreen and Alalfi, 2020b; Xu et al., 2020; Zhang et al., 2020a

### Manticore (2017-02, open source)

Dynamic symbolic execution framework for smart contract bytecode analysis

*Primary study introducing the tool:* Mossberg et al., 2019

*Discussed in 12 surveys:* Hu et al., 2021a; Kim and Ryu, 2020; Tolmach et al., 2020; Ferreira et al., 2020a; di Angelo and Salzer, 2019b; Durieux et al., 2020; Huang et al., 2019; López Vivar et al., 2020; Miller et al., 2018; Sayeed et al., 2020; Xu et al., 2020; Ye et al., 2019a

### ModCon (2020-11)

Model-based Solidity smart contract test generation and analysis, based on user-specified models and test-oracles.
`https://sites.google.com/view/modcon` (Souce code not available)

*Primary study introducing the tool:* Liu et al., 2020a

### *(NuSMV) Model checking* (2018-07)

Solidity smart contract model-checking approach, based on NuSMV model language translation and temporal logic proposition specification

*Primary study introducing the tool:* Nehaï et al., 2018

*Discussed in 4 surveys:* Hu et al., 2021a; Tolmach et al., 2020; Almakhour et al., 2020b,a

### *(*PROMELA *and* SPIN*) Model checking* (2020-02)

Smart contract model-checking, based on Solidity syntax and semantics formalization for translation to PROMELA models and subsequent application of the SPIN model checker

*Primary study introducing the tool:* Osterland and Rose, 2020

*Discussed in 1 survey:* Tolmach et al., 2020

### MOPS (2019-10)

Performance optimizing expansion of Mythril, identifying critical paths with potential vulnerabilities based on a multi-objective oriented path search (MOPS) strategy, guiding dynamic symbolic execution to improve efficiency. Additional security rules and vulnerability detection logics are introduced

*Primary study introducing the tool:* Fu et al., 2019

### MPro (2019-10)

Solidity smart contract analysis for deepth-n vulnerability detection, based on combining symbolic execution and data dependency analysis, built on Mythril and Slither

*Primary study introducing the tool:* Zhang et al., 2019b

### *MSgram analysis* (2020-07)

Data driven smart contract analysis for vulnerability detection, based on Multi-Semantic gram (MSgram), a variant of S-gram, by analyzing smart contract sequences of multiple semantics, covering AST representation and text information

*Primary study introducing the tool:* Yang et al., 2020a

### MuSC (2020-03, open source)

Solidity smart contract mutation testing demo for vulnerability detection supporting fast mutant generation, test net creation as well as test deployment and execution

*Primary study introducing the tool:* Li et al., 2019

*Discussed in 1 survey:* Kim and Ryu, 2020

### Mythril (2017-10, open source)

Smart contract vulnerability detection, based on a combination of various analysis methods such as symbolic execution, SMT solving and taint analysis

*Primary study introducing the tool:* Mueller, 2018

*Discussed in 18 surveys:* Hu et al., 2021a; Kim and Ryu, 2020; Singh et al., 2020; Tolmach et al., 2020; Ferreira et al., 2020a; Almakhour et al., 2020b; Chen et al., 2020a; di Angelo and Salzer, 2019b; Dika and Nowostawski, 2018; Durieux et al., 2020; Huang et al., 2019; López Vivar et al., 2020; Miller et al., 2018; Samreen and Alalfi, 2020b; Sayeed et al., 2020; Xu et al., 2020; Ye et al., 2019a; Zhang et al., 2020a

*Mythril extension for 'gasless send' issue detection* (2019-06)

Upgrade of Mythril analysis engine to allow for gas usage modeling

*Primary study introducing the tool:* Prechtel et al., 2019

*Discussed in 1 survey:* Tolmach et al., 2020

MythX (2018)

Mythril extension and implementation as proprietary cloud based smart contract security analysis API SaaS, supporting purpose-built security tool integration.
`https://mythx.io/`, `https://github.com/b-mueller/awesome-mythx-smart-contract-s`

*Discussed in 2 surveys:* Sayeed et al., 2020; Ye et al., 2019a

NeuCheck (2019-01, open source)

Syntax analysis, based on source code to intermediate representation transformation for security vulnerability detection

*Primary study introducing the tool:* Lu et al., 2019

*Discussed in 1 survey:* Hu et al., 2021a

NPCHECKER (2019-10)

'Nondeterministic Payment Checker' using static analysis to detect payment vulnerabilities, based on intermediate representation translation and instrumentation, information flow analysis and taint tracking techniques

*Primary study introducing the tool:* Wang et al., 2019b

*Discussed in 1 survey:* Tolmach et al., 2020

Octopus (2018-10, open source)

Smart contract security analysis framework for various blockchain platforms, including Bitcoin, EOS, NEO and Ethereum, capable of Ethereum WebAssembly (WASM) and bytecode analysis applying different methods, such as disassembly and IR translation, control and call flow analysis, or symbolic execution

*Discussed in 3 surveys:* Durieux et al., 2020; Huang et al., 2019; Ye et al., 2019a

Osiris (2018-09, open source)

Oyente based security analysis for integer bug detection in smart contracts combining symbolic execution with taint analysis

*Primary study introducing the tool:* Torres et al., 2018

*Discussed in 10 surveys:* Hu et al., 2021a; Tolmach et al., 2020; Ferreira et al., 2020a; Almakhour et al., 2020b; di Angelo and Salzer, 2019b; Durieux et al., 2020; Garfatta et al., 2021; Khan and Namin, 2020; López Vivar et al., 2020; Zhang et al., 2020a

### Oyente (2016-01, open source)

One of the first and longest maintained smart contract vulnerability detection tools, mainly based on symbolic execution

*Primary study introducing the tool:* Luu et al., 2016

*Discussed in 24 surveys:* Hu et al., 2021a; Kim and Ryu, 2020; Singh et al., 2020; Tolmach et al., 2020; Vacca et al., 2020; Ferreira et al., 2020a; Almakhour et al., 2020b; Chen et al., 2020a; di Angelo and Salzer, 2019b; Dika and Nowostawski, 2018; Durieux et al., 2020; Garfatta et al., 2021; Huang et al., 2019; Khan and Namin, 2020; López Vivar et al., 2020; Miller et al., 2018; Moona and Mathew, 2021; Perez and Livshits, 2019; Praitheeshan et al., 2020b; Samreen and Alalfi, 2020b; Sayeed et al., 2020; Xu et al., 2020; Ye et al., 2019a; Zhang et al., 2020a

### PASO (2019-09, open source)

Web based parser providing sofware metrics for Solidity smart contracts.
Live deployment: `https://aphd.github.io/paso/`

*Primary study introducing the tool:* Antonio Pierro and Tonelli, 2020

*Discussed in 1 survey:* Vacca et al., 2020

### *Payoff analyzer* (2018-04)

Game theoretic analysis, based on a specific, simplified programming language for smart contracts translation to state-based models and abstraction refinement to compute the worst-case guaranteed utilities, i.e. expected payoff and derive possible vulnerabilites, if the payoff lies outside of an expected range

*Primary study introducing the tool:* Chatterjee et al., 2018

*Discussed in 1 survey:* Kim and Ryu, 2020

### *Petri Nets based secure smart contract generation* (2020-01)

Prototype for design, development, and verification of secure Solidity smart contracts, consisting of a Petri Nets based visual modeling engine, a transition execution and simulation, a verification and validation engine and smart contract generation from modeled Petri Nets workflows

*Primary study introducing the tool:* Zupan et al., 2020

### *RA (Reentrancy Analyzer)* (2020-02, open source)

Static analyzer for reentrancy vulnerabilities on smart contract bytecode, combining symbolic execution and SMT solving

*Primary study introducing the tool:* Chinen et al., 2020

*Discussed in 1 survey:* Vacca et al., 2020

### Rattle (2018-08, open source)

Static smart contract binary analysis, based on generating a control flow graph and lifting it to Single Static Assignment (SSA) form and producing a graphical representation of the register machine

*Primary study introducing the tool:* Stortz, 2018

*Discussed in 3 surveys:* di Angelo and Salzer, 2019b; Durieux et al., 2020; López Vivar et al., 2020

### *Reentrancy detection* (2018-07)

Analysis framework combining static and dynamic analysis for reentrancy vulnerability detection in Solidity smart contracts, including the generation and execution of attack contracts to confirm exploitability and reduce false positive results

*Primary study introducing the tool:* Samreen and Alalfi, 2020a

*Discussed in 1 survey:* Vacca et al., 2020

### ReGuard (2018-05)

Dynamic analyzer for reentrancy vulnerability detection in Solidity smart contracts based on IR transformation, fuzzing and trace analysis

*Primary study introducing the tool:* Liu et al., 2018a

*Discussed in 10 surveys:* Hu et al., 2021a; Kim and Ryu, 2020; Tolmach et al., 2020; Vacca et al., 2020; Almakhour et al., 2020b; Chen et al., 2020a; di Angelo and Salzer, 2019b; Durieux et al., 2020; Khan and Namin, 2020; Samreen and Alalfi, 2020b

### RegularMutator (2020-09)

Mutation testing tool for Solidity smart contracts, including language-specific operators corresponding to common development errors

*Primary study introducing the tool:* Ivanova and Khritankov, 2020

### Remix-IDE (2014-11, open source)

Plugin architecture based Solidity smart contract development framework, including various debugging, unit testing and vulnerability detection modules.
`https://remix-ide.readthedocs.io/en/latest/`, `https://remix.ethereum.org/`

*Discussed in 5 surveys:* di Angelo and Salzer, 2019b; Dika and Nowostawski, 2018; Durieux et al., 2020; López Vivar et al., 2020; Zhang et al., 2020a

### SAFEVM (2019-07, open source)

Framework for Solidity and bytecode smart contracts verification, confirming general safety annotations and array access, based on Oyente, EthIR and C program verification engines.
Live deployment: `https://costa.fdi.ucm.es/gastap/`

*Primary study introducing the tool:* Albert et al., 2019a

*Discussed in 3 surveys:* Hu et al., 2021a; Kim and Ryu, 2020; Tolmach et al., 2020

### SASC (2018-02)

Security risk detection, based on symbolic execution, syntax analysis and topology diagram generation for manual analysis support

*Primary study introducing the tool:* Zhou et al., 2018a

*Discussed in 6 surveys:* Hu et al., 2021a; Singh et al., 2020; Tolmach et al., 2020; di Angelo and Salzer, 2019b; Durieux et al., 2020; Garfatta et al., 2021

### sCompile (2019-11)

Critical program path detection and risk priorisation for further analysis support in smart contracts based on CFGs, symbolic execution and SMT solving

*Primary study introducing the tool:* Chang et al., 2019

*Discussed in 6 surveys:* Hu et al., 2021a; Kim and Ryu, 2020; Chen et al., 2020a; di Angelo and Salzer, 2019b; Durieux et al., 2020; Khan and Namin, 2020

### SCREPAIR (2019-12, open source)

Automatic Solidity smart contract gas-optimization and vulnerability patch generation, based on mutation analysis and application of Slither and Oyente for vulnerability detection

*Primary study introducing the tool:* Yu et al., 2020

### Securify (2018-09, open source)

Security analysis and automated vulnerability detection tool for Solidity and bytecode smart contracts, based on a decompiler that symbolically encodes the dependency graph, abstract interpretation and compliance and violation pattern checks that can be extended and are specified in a domain-specific language

*Primary study introducing the tool:* Tsankov et al., 2018; Tsankov, 2018

*Discussed in 24 surveys:* Hu et al., 2021a; Kim and Ryu, 2020; Singh et al., 2020; Tolmach et al., 2020; Vacca et al., 2020; Ferreira et al., 2020a; Almakhour et al., 2020b; Chen et al., 2020a; di Angelo and Salzer, 2019b; Dika and Nowostawski, 2018; Durieux et al., 2020; Garfatta et al., 2021; Huang et al., 2019; Khan and Namin, 2020; López Vivar et al., 2020; Miller et al., 2018; Moona and Mathew, 2021; Perez and Livshits, 2019; Praitheeshan et al., 2020b; Samreen and Alalfi, 2020b; Sayeed et al., 2020; Xu et al., 2020; Ye et al., 2019a; Zhang et al., 2020a

### Seraph (2020-06)

EVM and WASM cross-platform security analysis, based on CFG generation, light weight symbolic execution and symbolic semantic graph (SSG) analysis.
Video: `https://youtu.be/wxixZkVqUsc`

*Primary study introducing the tool:* Yang et al., 2020c

### Sereum (2018-12)

Complex reentrancy vulnerability detection and prevention, such as cross-function, delegated and create-based reentrancy based on dynamic taint tracking and data-flow analysis.
`https://github.com/uni-due-syssec/eth-reentrancy-attack-patterns`

*Primary study introducing the tool:* Rodler et al., 2018

*Discussed in 4 surveys:* Hu et al., 2021a; Kim and Ryu, 2020; Chen et al., 2020a; Khan and Namin, 2020

### SERVOIS (2018-04, open source)

Automated generation of commutativity conditions and detection of concurrency-related vulnerabilities in smart contracts

*Primary study introducing the tool:* Bansal et al., 2018

*Discussed in 1 survey:* Kim and Ryu, 2020

### sFuzz (2020-06, open source)

Dynamic smart contract analysis, based on AFL, a fuzzer for C programs, applying a lightweight multi-objective adaptive strategy to cover hard to reach branches and achieve high code coverage for vulnerability detection

*Primary study introducing the tool:* Nguyen et al., 2020

### SIF (2019-09, open source)

Solidity smart contract analysis framework, facilitating tool integration, including source code level techniques, such as instrumentation, assertion checker to detect arithmetic vulnerabilities, AST and CFG generation, diagnostics and optimization for contract development support.
Live deployment: `https://wandbox.org/permlink/PnaL6bO9zipKRuKu`

*Primary study introducing the tool:* Peng et al., 2019

*Discussed in 2 surveys:* Kim and Ryu, 2020; Vacca et al., 2020

### Slither (2018-10, open source)

Static Solidity smart contract vulnerability detection including code optimization suggestions, based on intermediate language translation using Static Single Assignment (SSA) and graphical code visualization to support manual analysis

*Primary study introducing the tool:* Feist et al., 2019

*Discussed in 11 surveys:* Hu et al., 2021a; Kim and Ryu, 2020; Tolmach et al., 2020; Vacca et al., 2020; Ferreira et al., 2020a; Almakhour et al., 2020b; Durieux et al., 2020; López Vivar et al., 2020; Sayeed et al., 2020; Ye et al., 2019a; Zhang et al., 2020a

### Smart Contract Analyzer (2020-01, open source)

Solidity smart contract analysis and vulnerability detection based on AST generation

*Primary study introducing the tool:* Hwang and Ryu, 2020

### *Smart contract modeling and behavior verification* (2018-02)

Formal modeling approach for smart contract behavior verification, based on simulation in an execution environment and statistical model checking of behaviors revealing attack and breach scenarios

*Primary study introducing the tool:* Abdellatif and Brousmiche, 2018

*Discussed in 5 surveys:* Kim and Ryu, 2020; Tolmach et al., 2020; Almakhour et al., 2020b,a; Huang et al., 2019

### SmartAnvil/SmartShackle (2018-12, open source)

Tool chain framework for smart contract analysis comprising of different tools with various analysis techniques and approaches, such as AST generation, gas analysis, graph generation, vulnerability detection, query language for blockchain information retrieval

*Primary study introducing the tool:* Ducasse et al., 2019

### SmartBugs (2019-10, open source)

Extendable execution framework, facilitating the integration and comparison of multiple Solidity smart contract analysis tools, including a Docker images plugin system to simply add new tools, parallel execution and normalized output for easier result comparison and processing. Currently 11 tools are supported: Conkas, HONEYBADGER, MAIAN, Manticore, Mythril, Osiris, Oyente, Securify, Slither, SmartCheck and Solhint

*Primary study introducing the tool:* Ferreira et al., 2020b

*Discussed in 2 surveys:* Ferreira et al., 2020a; Durieux et al., 2020

### SmartCheck (2018-05, open source)

Static Solidity smart contract analysis for vulnerability detection checking XPath patterns on XML intermediate representation

*Primary study introducing the tool:* Tikhomirov et al., 2018

*Discussed in 19 surveys:* Hu et al., 2021a; Kim and Ryu, 2020; Vacca et al., 2020; Ferreira et al., 2020a; Almakhour et al., 2020b; Chen et al., 2020a; di Angelo and Salzer, 2019b; Dika and Nowostawski, 2018; Durieux et al., 2020; Huang et al., 2019; Khan and Namin, 2020; López Vivar et al., 2020; Miller et al., 2018; Moona and Mathew, 2021; Perez and Livshits, 2019; Samreen and Alalfi, 2020b; Sayeed et al., 2020; Ye et al., 2019a; Zhang et al., 2020a

### SmartEmbed (2019-05, open source)

Smart contract validation, vulnerability and clone detection, based on automated deep learning on structural Solidity code embedding and reporting code fragments similar to known vulnerabilities or clones

*Primary study introducing the tool:* Gao et al., 2019a; Gao, 2020; Gao et al., 2020, 2019c

*Discussed in 2 surveys:* Kim and Ryu, 2020; Vacca et al., 2020

### SmartInspect (2018-08, open source)

Part of the SmartAnvil/SmartShackle tool chain, analyzing smart contract states with decompilation, AST visualization and a mirror based architecture that mimics the contract structure to access the memory layout

*Primary study introducing the tool:* Bragagnolo et al., 2018

*Discussed in 1 survey:* Vacca et al., 2020

### SMARTSCOPY/SOLAR (2019-02)

Synthesizer of attack contracts that exploit smart contract vulnerabilities, introducing summary-based symbolic evaluation and optimizations to partition the synthesis search space for parallel exploration to improve efficiency and accuracy

*Primary study introducing the tool:* Feng et al., 2019b, 2020

*Discussed in 3 surveys:* Hu et al., 2021a; Kim and Ryu, 2020; Khan and Namin, 2020

### SMARTSHIELD (2020-02)

Automatic smart contract protection system against three vulnerabilities, namely state changes after external calls, missing checks for out-of-bound arithmetic operations, and missing checks for failing external calls

*Primary study introducing the tool:* Zhang et al., 2020d

*Discussed in 1 survey:* Samreen and Alalfi, 2020b

### *SMT-based verification* (2018-10)

SMT based formal verification module for integration with the Solidity compiler to automatically warn of potential vulnerabilities such as arithmetic overflow/underflow, unreachable code and assertion fails

*Primary study introducing the tool:* Alt and Reitwiessner, 2018

*Discussed in 2 surveys:* Hu et al., 2021a; Tolmach et al., 2020

### SODA (2020-03, open source)

Smart contract Online Detection framework against Attacks, supporting the development of detection tools and analysis of smart contract attacks on the blockchain, based on EVM instrumentation

*Primary study introducing the tool:* Chen et al., 2020c

*Discussed in 1 survey:* Tolmach et al., 2020

### SolAnalyser (2019-12)

Solidity smart contract vulnerability detection using both static and dynamic approaches, based on instrumentation and mutation

*Primary study introducing the tool:* Akca et al., 2019

### (Li and Long's) SOLAR (2019-02)

Detection of violation errors from smart contract standards, such as ERC-20 and ERC-721 standards, based on an optimized symbolic execution engine built on Manticore, behavior modeling as a state machine and SMT solving

*Primary study introducing the tool:* Li and Long, 2019

*Discussed in 1 survey:* Kim and Ryu, 2020

### SOLC-VERIFY (2019-07, open source)

Solidity smart contract verification, implemented as add-on to the Solidity compiler, based on AST transformation, translation to an intermediate representation and SMT solving

*Primary study introducing the tool:* Hajdu and Jovanović, 2020; Hajdu et al., 2020

*Discussed in 2 surveys:* Kim and Ryu, 2020; Tolmach et al., 2020

### Solgraph (2016-07, open source)

Solidity smart contract function control flow graph visualization supporting manual security analysis and potential vulnerability detection

*Discussed in 4 surveys:* di Angelo and Salzer, 2019b; Durieux et al., 2020; López Vivar et al., 2020; Miller et al., 2018

### Solhint (2017-10, open source)

Solidity smart contract linter, supporting security and style guide validations

*Discussed in 2 surveys:* Ferreira et al., 2020a; Durieux et al., 2020

### SoliAudit (2019-01, open source)

Smart contract vulnerability detection applying static and dynamic analysis approaches, based on fuzz testing and machine learning

*Primary study introducing the tool:* Liao et al., 2019

### Solicitous (2020-03, open source)

Solidity smart contract modeling using constrained Horn clauses to enable fully automated verification of safety properties

*Primary study introducing the tool:* Marescotti et al., 2020

*Discussed in 2 surveys:* Hu et al., 2021a; Tolmach et al., 2020

### SolidiFI (2020-05, open source)

Framework for automated evaluation of static smart contract analysis tools, based on Solidity smart contract AST analysis and vulnerability injection to establish a ground truth

*Primary study introducing the tool:* Ghaleb and Pattabiraman, 2020

### Solidifier (2020-02)

Bounded model checker for Solidity, based on a over-approximated formalization of Solidity and Ethereum, intermediate verification language translation, with the goal of finding program errors or bad states

*Primary study introducing the tool:* Antonino and Roscoe, 2020

*Discussed in 1 survey:* Tolmach et al., 2020

### SolidiKeY (2020-05, open source)

Prototype verification tool for invariant-based specification and verification, based on the deductive verification system KeY

*Primary study introducing the tool:* Ahrendt and Bubel, 2020

### SolidityCheck (2019-11, open source)

Static Solidity smart contract problem detection, including reentrancy and integer overflow vulnerabilities, based on regular expressions and program instrumentation

*Primary study introducing the tool:* Zhang et al., 2019a

*Discussed in 1 survey:* Zhang et al., 2020a

### SolMet (2018-08, open source)

Object oriented metrics calculation, such as number of source code lines, logical code lines, comment lines, functions, library or interfaces, including the weighted complexity, the sum of the deepest nesting level of the control structures, by parsing Solidity smart contracts

*Primary study introducing the tool:* Hegedus, 2018

*Discussed in 4 surveys:* Vacca et al., 2020; di Angelo and Salzer, 2019b; Durieux et al., 2020; López Vivar et al., 2020

### Solythesis (2020-02, open source)

Solidity smart contract source to source compilation and runtime validation, based on instrumented contract generation with user specified invariants, which reject all transactions violating the invariants

*Primary study introducing the tool:* Li et al., 2020

*Discussed in 1 survey:* Tolmach et al., 2020

### STAN (2020-12)

Description generator for smart contract bytecode, to support manual analysis, reverse engineering and readability of smart contract behavior, based on symbolic execution and NLP (Natural Language Processing)

*Primary study introducing the tool:* Li et al., 2020

### Syrup (2020-05, open source)

Gas super-optimization of smart contracts, based on symbolic execution and Max-SMT encoding, which finds bytecode blocks with minimal gas cost

*Primary study introducing the tool:* Albert et al., 2020b

*Discussed in 1 survey:* Hu et al., 2021a

### teEther (2019-02, open source)

Automated vulnerability detection and exploit generation on smart contract bytecode, based on CFG generation, symbolic execution and SMT solving

*Primary study introducing the tool:* Krupp and Rossow, 2018

*Discussed in 11 surveys:* Hu et al., 2021a; Kim and Ryu, 2020; Tolmach et al., 2020; Vacca et al., 2020; Chen et al., 2020a; di Angelo and Salzer, 2019b; Durieux et al., 2020; Huang et al., 2019; Khan and Namin, 2020; Perez and Livshits, 2019; Xu et al., 2020

### *Transaction-based analysis with LSTM network* (2021-03)

Transaction-based classification of Ethereum smart contracts for anomaly detection and malicious contract identification, using a long short-term memory (LSTM) network machine learning approach

*Primary study introducing the tool:* Hu et al., 2021b

### Vandal (2016-08, open source)

Smart contract bytecode security analysis framework for vulnerability detection, based on semantic logic relation conversion, CFG and data flow analysis, user generated security specifications and vulnerability queries

*Primary study introducing the tool:* Brent et al., 2018

*Discussed in 14 surveys:* Hu et al., 2021a; Kim and Ryu, 2020; Tolmach et al., 2020; Almakhour et al., 2020b; Chen et al., 2020a; di Angelo and Salzer, 2019b; Durieux et al., 2020; Garfatta et al., 2021; Khan and Namin, 2020; López Vivar et al., 2020; Perez and Livshits, 2019; Praitheeshan et al., 2020b; Samreen and Alalfi, 2020b; Sayeed et al., 2020

### VeriSmart (2020-04, open source)

Arithmetic safety verification for Solidity smart contracts, based on CEGIS-style verification, automatically inferring transaction invariants, providing safety property guarantees, such as freedom from integer overflow

*Primary study introducing the tool:* So et al., 2020

*Discussed in 3 surveys:* Hu et al., 2021a; Kim and Ryu, 2020; Tolmach et al., 2020

### Vertigo (2019-08, open source)

Mutation testing framework for Solidity smart contracts and Truffle, implementing a range of mutation operators

*Primary study introducing the tool:* Honig et al., 2019

### VerX (2020-05)

Automatic verification to prove functional properties and temporal safety properties, based on reachability checking, code instrumentation, symbolic execution and delayed predicate abstraction.
`https://verx.ch/`

*Primary study introducing the tool:* Permenev et al., 2020

*Discussed in 2 surveys:* Hu et al., 2021a; Vacca et al., 2020

### Visualgas (2018-11)

Gas cost analysis for Solidity smart contracts, based on AST generation, fuzz testing and transaction trace analysis to support gas-efficient smart contract development

*Primary study introducing the tool:* Signer, 2018

### VulDeeSmartContract (2020-01, open source)

Reentrancy vulnerability detection in Solidity smart contracts, based on an extension of the VulDeePecker framework implementing the bidirectional long-short term memory with attention mechanism (BLSTM-ATT) deep learning approach

*Primary study introducing the tool:* Qian et al., 2020

### ZEUS (2018-02)

Smart contract security verification framework and vulnerability detection, based on Solidity to LLVM translation, abstract interpretation and symbolic model checking via constrained horn clauses

*Primary study introducing the tool:* Kalra et al., 2018

*Discussed in 19 surveys:* Hu et al., 2021a; Kim and Ryu, 2020; Singh et al., 2020; Tolmach et al., 2020; Vacca et al., 2020; Almakhour et al., 2020b,a; Chen et al., 2020a; di Angelo and Salzer, 2019b; Durieux et al., 2020; Garfatta et al., 2021; Huang et al., 2019; Khan and Namin, 2020; Miller et al., 2018; Moona and Mathew, 2021; Perez and Livshits, 2019; Praitheeshan et al., 2020b; Samreen and Alalfi, 2020b; Sayeed et al., 2020

## 2.2 Open Source Tools

For the tools with open source, we additionally indicate in table S1 a URL to the respective repository, the publication date and last update if any, as well as the programming language(s) for the implementation.

Table S1: Information on open source tools

| Tool | Publ. date | Last update | Open source reference | Source language |
|---|---|---|---|---|
| ÆGIS | 2020-08 | ✗ | `https://github.com/christoftorres/Aegis` | Python |
| ConCert | 2020-01 | ✗ | `https://github.com/AU-COBRA/ConCert/tree/artefact` | Coq |
| Conkas | 2021-03 | ✗ | `https://github.com/nveloso/conkas` | Python |
| *(Dr. Y's Ethereum) Contract Analyzer* | 2016-09 | 2017-07 | `https://github.com/pirapira/dry-analyzer` | Coq, Ocaml |
| ContractFuzzer | 2018-08 | 2020-03 | `https://github.com/gongbell/ContractFuzzer` | Go, JavaScript |
| ContractLarva | 2019-08 | 2021-03 | `https://github.com/gordonpace/contractLarva` | Haskell |
| ContractMut | 2020-03 | ✗ | `https://doi.org/10.5281/zenodo.3726690` | JavaScript |
| ContraMaster / Vultron | 2019-05 | 2019-11 | `https://github.com/ntu-SRSLab/vultron` | JavaScript |
| DappGuard | 2017-05 | ✗ | `https://github.com/cookt/857final` | JavaScript, Python |
| DefectChecker | 2021-01 | ✗ | `https://github.com/Jiachi-Chen/DefectChecker/` | Java |
| E-EVM | 2018-01 | ✗ | `https://github.com/pisocrob/E-EVM` | Python |
| EASYFLOW | 2018-05 | 2018-08 | `https://github.com/Jianbo-Gao/EasyFlow/` | Go |
| ECF Checker | 2017-10 | ✗ | `https://github.com/shellygr/ECFChecker` | Go |
| Echidna | 2020-01 | 2021-07 | `https://github.com/crytic/echidna` | Haskell |
| Echidna-Parade | 2021-07 | ✗ | `https://github.com/crytic/echidna-parade` | Python |
| Erays | 2018-10 | ✗ | `https://github.com/teamnsrg/erays` | Python |

continued on next page

Table S1 (cont'd): Information on open source tools

| Tool | Publ. date | Last update | Open source reference | Source language |
|---|---|---|---|---|
| Ethainter | 2020-04 | ✗ | https://doi.org/10.5281/zenodo.3760402 | |
| ETHBMC | 2020-04 | 2021-06 | https://github.com/RUB-SysSec/EthBMC | Rust |
| Ethersplay | 2018-05 | 2021-07 | https://github.com/crytic/ethersplay | Python |
| EtherTrust | 2018-05 | 2019-08 | https://www.netidee.at/ethertrust https://github.com/SecPriv/EtherTrust | Java |
| EthIR | 2020-05 | 2021-01 | https://github.com/costa-group/EthIR | Python |
| eThor | 2020-10 | N/A | https://secpriv.wien/ethor/ | Java |
| ETHRACER | 2018-08 | 2019-05 | https://github.com/ashgeek/Ethracer | Python |
| *EVM Memory Modeling* | 2020-09 | ✗ | https://doi.org/10.5281/zenodo.4059797 | Python |
| *F* EVM small-step semantics* | 2018-04 | N/A | https://secpriv.tuwien.ac.at/tools/ethsemantics | F#, F* |
| FAIRCON | 2020-05 | 2020-08 | https://github.com/ntu-SRSLab/FairCon https://doi.org/10.1145/3410249 https://doi.org/10.21979/N9/0BEVRT | C++ |
| FSMC | 2019-05 | ✗ | https://github.com/d-suvorov/fsmc/ | Kotlin |
| FSolidM / VeriSolid Framework | 2017-10 | 2019-09 | https://github.com/anmavrid/smart-contracts https://cps-vo.org/group/SmartContracts | JavaScript |
| GASOL | 2020-02 | ✗ | https://doi.org/10.6084/m9.figshare.11876697.v1 | PHP, Python |
| Gigahorse | 2019-01 | 2021-07 | http://doi.org/10.5281/zenodo.2578692 https://github.com/nevillegrech/gigahorse-toolchain | Python |

continued on next page

Table S1 (cont'd): Information on open source tools

| Tool | Publ. date | Last update | Open source reference | Source language |
|---|---|---|---|---|
| GNNSCVulDetector | 2020-06 | 2021-03 | `https://github.com/Messi-Q/GNNSCVulDetector` | Python |
| HARVEY / BRAN | 2018-08 | 2020-01 | Partially open source: `https://github.com/Practical-Formal-Methods/bran` | Go |
| HONEYBADGER | 2019-02 | 2019-06 | `https://github.com/christoftorres/HoneyBadger` | Python |
| Hydra | 2017-11 | 2018-02 | `https://github.com/IC3Hydra/Hydra` | Haskell, Python |
| ILF | 2019-11 | 2020-10 | `https://github.com/eth-sri/ilf` | Python, Go |
| *Isabelle / HOL- based framework* | 2017-03 | 2018-04 | `https://github.com/pirapira/eth-isabelle` | Isabelle, OCaml |
| JAVADITY | 2018-07 | ✗ | `https://github.com/rebiscov/Javadity` | Java,ANTLR |
| KEVM | 2019-07 | 2021-07 | `https://github.com/kframework/evm-semantics` | Python, Java |
| *KEVM Verifier* | 2018-10 | 2021-04 | `https://github.com/runtimeverification/verified-smart-contracts` | Java |
| KSolidity | 2019-07 | ✗ | `https://github.com/kframework/solidity-semantics` | K |
| *LSTM Machine learning* | 2018-11 | 2019-06 | `https://github.com/wesleyjtann/Safe-SmartContracts` | Python |
| MadMax | 2018-09 | 2020-12 | `https://github.com/nevillegrech/MadMax` | Python |
| MAIAN | 2018-03 | ✗ | `https://github.com/ivicanikolicsg/MAIAN` | Python |
| Manticore | 2017-02 | 2021-07 | `https://github.com/trailofbits/manticore/` | Python |
| MuSC | 2020-03 | ✗ | `https://github.com/belikout/MuSC-Tool-Demo-repo` | JavaScript, Java |

continued on next page

Table S1 (cont'd): Information on open source tools

| Tool | Publ. date | Last update | Open source reference | Source language |
|---|---|---|---|---|
| Mythril | 2017-10 | 2020-11 | https://github.com/wflk/mythril https://github.com/b-mueller/mythril/ https://github.com/ConsenSys/mythril | Python |
| NeuCheck | 2019-01 | 2020-03 | https://github.com/Northeastern-University-Blockchain/NeuCheck | Java, ANTLR |
| Octopus | 2018-10 | 2020-11 | https://github.com/quoscient/octopus | Python |
| Osiris | 2018-09 | ✗ | https://github.com/christoftorres/Osiris | Python |
| Oyente | 2016-01 | 2020-11 | https://github.com/enzymefinance/oyente | Python, JavaScript |
| PASO | 2019-09 | 2021-07 | https://github.com/aphd/paso | JavaScript |
| *RA (Reentrancy Analyzer)* | 2020-02 | 2020-06 | https://github.com/wanidon/RA | Python |
| Rattle | 2018-08 | 2020-04 | https://github.com/crytic/rattle | Python |
| Remix-IDE | 2014-11 | 2021-07 | https://github.com/ethereum/remix-project | JavaScript |
| SAFEVM | 2019-07 | ✗ | http://costa.fdi.ucm.es/papers/costa/safevm.ova | Python |
| SCREPAIR | 2019-12 | 2020-05 | https://SCRepair-APR.github.io | C++ |
| Securify | 2018-09 | 2020-04 | https://github.com/eth-sri/securify https://github.com/eth-sri/securify2 | Java , Python |
| SERVOIS | 2018-04 | 2019-02 | https://github.com/kbansal/servois | Python |
| sFuzz | 2020-06 | 2020-07 | https://github.com/duytai/sFuzz | C++ |
| SIF | 2019-09 | 2020-06 | https://github.com/chao-peng/SIF | C++ |
| Slither | 2018-10 | 2021-07 | https://github.com/crytic/slither | Python |
| Smart Contract Analyzer | 2020-01 | ✗ | https://github.com/sjmini/icse2020-Solidity | Java |

continued on next page

Table S1 (cont'd): Information on open source tools

| Tool | Publ. date | Last update | Open source reference | Source language |
|---|---|---|---|---|
| SmartAnvil / SmartShackle | 2018-12 | 2019-11 | https://github.com/smartanvil | Smalltalk |
| SmartBugs | 2019-10 | 2021-07 | https://github.com/smartbugs/smartbugs | Python |
| SmartCheck | 2018-05 | 2020-11 | https://github.com/smartdec/smartcheck | Java |
| SmartEmbed | 2019-05 | 2020-08 | https://github.com/beyondacm/SmartEmbed | JavaScript, Java |
| SmartInspect | 2018-08 | ✗ | https://github.com/RMODINRIA-Blockchain/SmartShackle https://github.com/smartanvil/SmartShackle | Smalltalk |
| SODA | 2020-03 | 2020-09 | https://github.com/pandabox-dev/SODA | Go |
| SOLC-VERIFY | 2019-07 | 2020-07 | https://github.com/SRI-CSL/solidity/blob/0.7/SOLC-VERIFY-README.md | Python |
| Solgraph | 2016-07 | 2019-01 | https://github.com/raineorshine/solgraph | JavaScript |
| Solhint | 2017-10 | 2021-05 | https://github.com/protofire/solhint | JavaScript |
| SoliAudit | 2019-01 | ✗ | https://github.com/jianwei76/SoliAudit | JupyterNotebook, Python |
| Solicitous | 2020-03 | 2020-05 | https://github.com/usi-verification-and-security/solc | C++ |
| SolidiFI | 2020-05 | 2021-05 | https://github.com/DependableSystemsLab/SolidiFI | Python |
| SolidiKeY | 2020-05 | ✗ | https://www.key-project.org/isola2020-smart/ | Java |
| SolidityCheck | 2019-11 | 2021-06 | https://github.com/xf97/SolidityCheck | C++ |
| SolMet | 2018-08 | 2020-11 | https://github.com/chicxurug/SolMet-Solidity-parser | Java |
| Solythesis | 2020-02 | 2020-04 | https://github.com/Leeleo3x/solythesis-artifact | Rust |

continued on next page

Table S1 (cont'd): Information on open source tools

| Tool | Publ. date | Last update | Open source reference | Source language |
|---|---|---|---|---|
| Syrup | 2020-05 | ✗ | `https://github.com/mariaschett/syrup-backend` | OCaml, StandardML |
| teEther | 2019-02 | 2021-07 | `https://github.com/nescio007/teether` | Python |
| Vandal | 2016-08 | 2020-07 | `https://github.com/usyd-blockchain/vandal` | Python |
| VeriSmart | 2020-04 | 2020-05 | `https://github.com/kupl/VeriSmart-public` | OCaml |
| Vertigo | 2019-08 | 2021-02 | `https://github.com/JoranHonig/vertigo` | Python |
| VulDeeSmartContract | 2020-01 | 2020-05 | `https://github.com/Messi-Q/ReChecker` | Python |

## 2.3 Properties and Methods of Tools

The overview of the properties and methods presents the tools in alphabetical order in a multi-page table that we split in two parts. In part 1 of the table, we indicate the code level, aim, analysis type, and code transformation techniques. In part 2, we indicate the analysis methods that they tools employ.

Table S2: Characteristics of analysis tools – part 1

| Tool | EVM bytecode | Source code | Vulnerability detection | Program Correctness | Gas / Resource Analysis | Bulk Analysis / Full Automation | Exploit Generation | Reactive Defenses | Manual Analysis / Dev. Support | Static Analysis | Dynamic Analysis | Control Flow Graph (CFG) | Transaction Analysis / Traces | Abstract Syntax Tree (AST) | Decompilation | Interm. Rep. / Spec. Language | Disassembly | Finite State Machine (FSM) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ABBE | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ● | ● | ○ | ○ | ○ | ○ |
| ADF-GA | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ● | ● | ○ | ○ | ○ | ○ | ○ |
| ÆGIS | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Annotary | ● | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| CESC | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ● | ● | ○ | ○ | ○ | ○ | ○ |
| Clairvoyance | ○ | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ |
| *Comparable vector encoding and matching* | ● | ○ | ● | ○ | ○ | ○ | ○ | ● | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ |

continued on next page

Table S2 (cont'd): Characteristics of analysis tools – part 1

| Tool | EVM bytecode | Source code | Vulnerability detection | Program Correctness | Gas / Resource Analysis | Bulk Analysis / Full Automation | Exploit Generation | Reactive Defenses | Manual Analysis / Dev. Support | Static Analysis | Dynamic Analysis | Control Flow Graph (CFG) | Transaction Analysis / Traces | Abstract Syntax Tree (AST) | Decompilation | Interm. Rep. / Spec. Language | Disassembly | Finite State Machine (FSM) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ConCert | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ● | ○ | ● | ○ | ○ |
| Conkas | ● | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ |
| *(Dr. Y's Ethereum) Contract Analyzer* | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| ContractFuzzer | ● | ○ | ● | ○ | ○ | ○ | ● | ● | ○ | ○ | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ |
| ContractGuard (GuardStrike) | ○ | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ |
| ContractGuard (IDS) | ○ | ● | ● | ○ | ○ | ○ | ○ | ○ | ● | ● | ● | ● | ● | ○ | ○ | ○ | ○ | ○ |
| ContractLarva | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ● |
| ContractMut | ○ | ● | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ● | ○ | ○ | ● | ○ | ○ | ○ | ○ |
| ContractVis | ● | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ |
| ContractWard | ○ | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| ContraMaster / Vultron | ○ | ● | ● | ○ | ○ | ● | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| *Convolutional neural network* | ● | ● | ○ | ○ | ● | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| DappGuard | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| *Deductive verification with Why3* | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ |
| DefectChecker | ● | ○ | ● | ○ | ○ | ○ | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Deviant | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ |
| E-EVM | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ● | ○ |
| EASYFLOW | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| ECFChecker | ● | ○ | ○ | ● | ○ | ○ | ○ | ○ | ● | ● | ● | ○ | ● | ○ | ○ | ● | ○ | ○ |
| Echidna | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Echidna-Parade | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Erays | ● | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ● | ○ |
| Ethainter | ● | ○ | ● | ○ | ○ | ● | ○ | ● | ○ | ○ | ● | ○ | ● | ○ | ○ | ○ | ● | ○ |
| ETHBMC | ● | ○ | ● | ● | ○ | ○ | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Ether* (S-GRAM) | ● | ● | ● | ○ | ○ | ○ | ○ | ● | ○ | ● | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ |
| Etherolic | ● | ○ | ● | ○ | ○ | ● | ○ | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Ethersplay | ● | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ● | ○ |
| EtherTrust | ● | ○ | ● | ● | ○ | ○ | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| EthIR | ● | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ● | ○ | ● | ● | ○ | ○ | ○ | ● | ○ |
| eThor | ● | ○ | ● | ● | ○ | ○ | ○ | ● | ○ | ● | ○ | ● | ● | ○ | ○ | ● | ○ | ○ |
| ETHPLOIT | ○ | ● | ● | ○ | ○ | ● | ○ | ● | ○ | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ |

continued on next page

Table S2 (cont'd): Characteristics of analysis tools – part 1

| Tool | EVM bytecode | Source code | Vulnerability detection | Program Correctness | Gas / Resource Analysis | Bulk Analysis / Full Automation | Exploit Generation | Reactive Defenses | Manual Analysis / Dev. Support | Static Analysis | Dynamic Analysis | Control Flow Graph (CFG) | Transaction Analysis / Traces | Abstract Syntax Tree (AST) | Decompilation | Interm. Rep. / Spec. Language | Disassembly | Finite State Machine (FSM) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ETHRACER | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| EthScope | ● | ○ | ● | ○ | ○ | ○ | ○ | ● | ○ | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| *EVM Memory Modeling* | ● | ○ | ● | ○ | ● | ○ | ● | ○ | ○ | ● | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ |
| EVM* | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ |
| *F* EVM small-step semantics* | ● | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ |
| *F* Verification* | ● | ● | ● | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ |
| FAIRCON | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ |
| FSFC | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| FSMC | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● |
| FSolidM / VeriSolid Framework | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● |
| FSPVM-E / FEther | ○ | ● | ○ | ● | ○ | ○ | ○ | ● | ○ | ● | ● | ○ | ○ | ○ | ○ | ● | ○ | ○ |
| GasChecker | ● | ○ | ○ | ○ | ○ | ○ | ● | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| GASOL | ● | ● | ○ | ○ | ○ | ○ | ● | ○ | ○ | ● | ○ | ● | ○ | ○ | ● | ● | ○ | ○ |
| Gasper | ● | ○ | ○ | ○ | ○ | ○ | ● | ● | ○ | ● | ○ | ● | ○ | ○ | ● | ○ | ● | ○ |
| GasReducer | ● | ○ | ○ | ○ | ○ | ○ | ● | ● | ○ | ● | ○ | ● | ● | ○ | ● | ○ | ● | ○ |
| GASTAP | ● | ● | ○ | ○ | ○ | ○ | ● | ○ | ○ | ● | ○ | ● | ○ | ○ | ● | ○ | ○ | ○ |
| Gazfuzzer | ● | ○ | ● | ○ | ○ | ○ | ● | ○ | ○ | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Gigahorse | ● | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ● | ○ | ● | ○ | ○ | ● | ● | ○ | ○ |
| GNNSCVulDetector | ○ | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ● | ● | ○ | ○ | ○ | ○ | ○ |
| HARVEY / BRAN | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ● | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ |
| HONEYBADGER | ● | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Hydra | ○ | ● | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| ILF | ○ | ● | ● | ○ | ○ | ○ | ○ | ● | ○ | ● | ● | ● | ○ | ○ | ○ | ○ | ● | ○ |
| *Isabelle/HOL-based framework* | ● | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| JAVADITY | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ |
| KEVM | ● | ○ | ○ | ● | ○ | ○ | ● | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ |
| *KEVM Verifier* | ● | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ |
| KSolidity | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ |
| *LSTM Machine learning* | ● | ○ | ● | ○ | ○ | ○ | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| MadMax | ● | ○ | ● | ○ | ○ | ○ | ● | ● | ○ | ● | ○ | ● | ○ | ○ | ● | ○ | ○ | ○ |
| MAIAN | ● | ○ | ● | ○ | ○ | ● | ○ | ● | ○ | ● | ● | ○ | ● | ○ | ● | ○ | ○ | ○ |
| Manticore | ● | ○ | ● | ● | ○ | ● | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ |

continued on next page

Table S2 (cont'd): Characteristics of analysis tools – part 1

| Tool | EVM bytecode | Source code | Vulnerability detection | Program Correctness | Gas / Resource Analysis | Bulk Analysis / Full Automation | Exploit Generation | Reactive Defenses | Manual Analysis / Dev. Support | Static Analysis | Dynamic Analysis | Control Flow Graph (CFG) | Transaction Analysis / Traces | Abstract Syntax Tree (AST) | Decompilation | Interm. Rep. / Spec. Language | Disassembly | Finite State Machine (FSM) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ModCon | ○ | ● | ● | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| *(NuSMV) Model checking* | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| *(*PROMELA *and* SPIN*) Model checking* | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ |
| MOPS | ● | ● | ● | ○ | ○ | ○ | ○ | ● | ○ | ● | ● | ● | ○ | ○ | ○ | ○ | ● | ○ |
| MPro | ○ | ● | ● | ○ | ○ | ○ | ○ | ● | ○ | ● | ○ | ● | ○ | ○ | ○ | ● | ● | ○ |
| *MSgram analysis* | ○ | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ |
| MuSC | ○ | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Mythril | ● | ● | ● | ○ | ○ | ○ | ○ | ● | ○ | ● | ● | ● | ○ | ○ | ○ | ○ | ● | ○ |
| *Mythril extension for 'gasless send'* | ● | ● | ● | ○ | ● | ○ | ● | ● | ○ | ● | ● | ● | ○ | ○ | ○ | ○ | ● | ○ |
| MythX | ● | ● | ● | ○ | ● | ○ | ● | ○ | ○ | ● | ● | ● | ○ | ● | ○ | ● | ● | ○ |
| NeuCheck | ○ | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ |
| NPCHECKER | ● | ○ | ● | ● | ○ | ○ | ○ | ● | ○ | ● | ○ | ○ | ● | ○ | ○ | ● | ● | ○ |
| Octopus | ● | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ● | ○ | ● | ● | ○ | ○ | ● | ● | ○ |
| Osiris | ● | ● | ● | ○ | ○ | ○ | ○ | ● | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ |
| Oyente | ● | ● | ● | ○ | ○ | ○ | ○ | ● | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ● | ○ |
| PASO | ○ | ● | ○ | ○ | ● | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| *Payoff analyzer* | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ● |
| *Petri Nets based secure smart contract generation* | ○ | ● | ○ | ● | ● | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| *RA (Reentrancy Analyzer)* | ● | ○ | ● | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ |
| Rattle | ● | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ● | ○ |
| *Reentrancy detection* | ○ | ● | ● | ○ | ○ | ● | ○ | ○ | ○ | ● | ● | ○ | ○ | ● | ○ | ● | ○ | ○ |
| ReGuard | ○ | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ● | ● | ● | ○ | ● | ○ | ○ |
| RegularMutator | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Remix-IDE | ○ | ● | ● | ○ | ● | ○ | ● | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| SAFEVM | ● | ● | ○ | ● | ○ | ○ | ○ | ● | ○ | ● | ○ | ● | ○ | ○ | ○ | ● | ○ | ○ |
| SASC | ○ | ● | ● | ○ | ● | ○ | ○ | ● | ○ | ● | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ |
| sCompile | ● | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ● | ○ |
| SCREPAIR | ○ | ● | ● | ○ | ○ | ○ | ● | ○ | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Securify | ● | ● | ● | ● | ○ | ○ | ○ | ● | ○ | ● | ○ | ● | ● | ○ | ● | ○ | ○ | ○ |
| Seraph | ○ | ● | ● | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ |

continued on next page

Table S2 (cont'd): Characteristics of analysis tools – part 1

| Tool | EVM bytecode | Source code | Vulnerability detection | Program Correctness | Gas / Resource Analysis | Bulk Analysis / Full Automation | Exploit Generation | Reactive Defenses | Manual Analysis / Dev. Support | Static Analysis | Dynamic Analysis | Control Flow Graph (CFG) | Transaction Analysis / Traces | Abstract Syntax Tree (AST) | Decompilation | Interm. Rep. / Spec. Language | Disassembly | Finite State Machine (FSM) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Sereum | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ |
| SERVOIS | ○ | ● | ● | ○ | ● | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| sFuzz | ● | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ |
| SIF | ○ | ● | ● | ○ | ● | ○ | ○ | ○ | ○ | ● | ○ | ● | ○ | ● | ○ | ● | ○ | ○ |
| Slither | ○ | ● | ● | ○ | ● | ○ | ○ | ○ | ○ | ● | ○ | ● | ○ | ○ | ○ | ● | ○ | ○ |
| Smart Contract Analyzer | ○ | ● | ● | ○ | ○ | ○ | ○ | ● | ○ | ● | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ |
| *Smart contract modeling and behavior verification* | ○ | ● | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ● | ○ | ○ |
| SmartAnvil / SmartShackle | ● | ● | ● | ● | ● | ○ | ○ | ○ | ○ | ● | ○ | ● | ● | ● | ● | ● | ● | ○ |
| SmartBugs | ○ | ● | ● | ● | ● | ● | ○ | ● | ○ | ● | ● | ● | ● | ● | ● | ● | ● | ○ |
| SmartCheck | ○ | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ● | ○ | ● | ○ | ● | ○ | ○ |
| SmartEmbed | ○ | ● | ● | ○ | ○ | ○ | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| SmartInspect | ○ | ● | ● | ○ | ● | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ● | ● | ○ | ○ | ○ |
| SMARTSCOPY / SOLAR | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| SMARTSHIELD | ● | ○ | ● | ● | ○ | ○ | ○ | ● | ● | ○ | ● | ● | ● | ● | ○ | ● | ○ | ○ |
| *SMT-based verification* | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ |
| SODA | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| SolAnalyser | ○ | ● | ● | ○ | ○ | ○ | ○ | ● | ○ | ● | ● | ○ | ○ | ● | ○ | ○ | ○ | ○ |
| SOLAR | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ● |
| SOLC-VERIFY | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ● | ○ | ● | ○ | ○ |
| Solgraph | ○ | ● | ● | ○ | ● | ○ | ○ | ○ | ○ | ● | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ |
| Solhint | ○ | ● | ○ | ○ | ● | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| SoliAudit | ● | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Solicitous | ○ | ● | ○ | ● | ○ | ○ | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| SolidiFI | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ● | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ |
| Solidifier | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ |
| SolidiKeY | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| SolidityCheck | ○ | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| SolMet | ○ | ● | ○ | ○ | ● | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ |
| Solythesis | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| STAN | ● | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ |
| Syrup | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |

continued on next page

Table S2 (cont'd): Characteristics of analysis tools – part 1

| Tool | EVM bytecode | Source code | Vulnerability detection | Program Correctness | Gas / Resource Analysis | Bulk Analysis / Full Automation | Exploit Generation | Reactive Defenses | Manual Analysis / Dev. Support | Static Analysis | Dynamic Analysis | Control Flow Graph (CFG) | Transaction Analysis / Traces | Abstract Syntax Tree (AST) | Decompilation | Interm. Rep. / Spec. Language | Disassembly | Finite State Machine (FSM) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| teEther | ● | ○ | ● | ○ | ○ | ● | ○ | ● | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ● | ○ |
| *Transaction-based analysis with LSTM network* | ● | ○ | ● | ○ | ○ | ○ | ○ | ● | ● | ● | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ |
| Vandal | ● | ○ | ● | ● | ○ | ○ | ○ | ● | ○ | ● | ○ | ● | ● | ○ | ● | ● | ● | ○ |
| VeriSmart | ○ | ● | ○ | ● | ○ | ○ | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Vertigo | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| VerX | ● | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ |
| Visualgas | ○ | ● | ○ | ○ | ● | ○ | ● | ○ | ○ | ● | ○ | ○ | ● | ● | ○ | ○ | ○ | ○ |
| VulDeeSmartContract | ○ | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| ZEUS | ○ | ● | ● | ● | ○ | ○ | ○ | ● | ○ | ● | ○ | ○ | ○ | ● | ○ | ● | ○ | ○ |

Table S3: Characteristics of analysis tools – part 2

| Tool | Symbolic Execution | Formal Verification | Model Checking | Abstract Interpretation | Fuzzing | Runtime Verification | Concolic Testing | Mutation Testing | Pattern Matching | Code Instrumentation | Machine Learning | Taint Analysis |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ABBE | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ |
| ADF-GA | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ |
| ÆGIS | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Annotary | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ |
| CESC | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ |
| Clairvoyance | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● |
| *Comparable vector encoding and matching* | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ |
| ConCert | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Conkas | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |

continued on next page

Table S3 (cont'd): Characteristics of analysis tools – part 2

| Tool | Symbolic Execution | Formal Verification | Model Checking | Abstract Interpretation | Fuzzing | Runtime Verification | Concolic Testing | Mutation Testing | Pattern Matching | Code Instrumentation | Machine Learning | Taint Analysis |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *(Dr. Y's Ethereum) Contract Analyzer* | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| ContractFuzzer | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ● | ○ | ○ |
| ContractGuard (GuardStrike) | ● | ● | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| ContractGuard (IDS) | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ● | ○ | ○ |
| ContractLarva | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ● | ○ | ○ |
| ContractMut | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ |
| ContractVis | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| ContractWard | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ |
| ContraMaster / Vultron | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| *Convolutional neural network* | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ |
| DappGuard | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ |
| *Deductive verification with Why3* | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| DefectChecker | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Deviant | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ |
| E-EVM | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| EASYFLOW | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● |
| ECFChecker | ○ | ● | ● | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ |
| Echidna | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Echidna-Parade | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Erays | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Ethainter | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● |
| ETHBMC | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Ether* (S-GRAM) | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ |
| Etherolic | ● | ○ | ○ | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ● |
| Ethersplay | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| EtherTrust | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| EthIR | ● | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| eThor | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| ETHPLOIT | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ● | ○ | ● |
| ETHRACER | ● | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| EthScope | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ |
| *EVM Memory Modeling* | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| EVM* | ○ | ● | ○ | ○ | ○ | ● | ○ | ○ | ○ | ● | ○ | ○ |
| *F* EVM small-step semantics* | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |

continued on next page

Table S3 (cont'd): Characteristics of analysis tools – part 2

| Tool | Symbolic Execution | Formal Verification | Model Checking | Abstract Interpretation | Fuzzing | Runtime Verification | Concolic Testing | Mutation Testing | Pattern Matching | Code Instrumentation | Machine Learning | Taint Analysis |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *F* Verification* | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| FAIRCON | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ |
| FSFC | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ |
| FSMC | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| FSolidM / VeriSolid Framework | ○ | ○ | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| FSPVM-E / FEther | ● | ● | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ |
| GasChecker | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ |
| GASOL | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ |
| Gasper | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ |
| GasReducer | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ |
| GASTAP | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ |
| Gazfuzzer | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Gigahorse | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| GNNSCVulDetector | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| HARVEY / BRAN | ○ | ○ | ○ | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| HONEYBADGER | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ |
| Hydra | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| ILF | ● | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ |
| *Isabelle/HOL-based framework* | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| JAVADITY | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| KEVM | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| *KEVM Verifier* | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| KSolidity | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| *LSTM Machine learning* | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ |
| MadMax | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| MAIAN | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Manticore | ● | ● | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ |
| ModCon | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| *(NuSMV) Model checking* | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| *(*PROMELA *and* SPIN*) Model checking* | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| MOPS | ● | ● | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ● |
| MPro | ● | ● | ○ | ○ | ○ | ○ | ● | ○ | ● | ○ | ○ | ● |
| *MSgram analysis* | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ |
| MuSC | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ |

continued on next page

Table S3 (cont'd): Characteristics of analysis tools – part 2

| Tool | Symbolic Execution | Formal Verification | Model Checking | Abstract Interpretation | Fuzzing | Runtime Verification | Concolic Testing | Mutation Testing | Pattern Matching | Code Instrumentation | Machine Learning | Taint Analysis |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Mythril | ● | ● | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ● |
| *Mythril extension for 'gasless send'* | ● | ● | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ● |
| MythX | ● | ● | ○ | ○ | ● | ○ | ● | ○ | ○ | ● | ○ | ○ |
| NeuCheck | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ |
| NPCHECKER | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ● |
| Octopus | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Osiris | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● |
| Oyente | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| PASO | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ |
| *Payoff analyzer* | ○ | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| *Petri Nets based secure smart contract generation* | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| *RA (Reentrancy Analyzer)* | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Rattle | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| *Reentrancy detection* | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ● | ● | ○ | ○ |
| ReGuard | ○ | ○ | ○ | ○ | ● | ● | ○ | ○ | ● | ○ | ○ | ○ |
| RegularMutator | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Remix-IDE | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ |
| SAFEVM | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| SASC | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ |
| sCompile | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| SCREPAIR | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ |
| Securify | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ |
| Seraph | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Sereum | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● |
| SERVOIS | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| sFuzz | ○ | ○ | ○ | ○ | ● | ○ | ○ | ● | ○ | ● | ○ | ○ |
| SIF | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ |
| Slither | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ |
| Smart Contract Analyzer | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ |
| *Smart contract modeling and behavior verification* | ○ | ● | ● | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ |
| SmartAnvil / SmartShackle | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ |
| SmartBugs | ● | ● | ○ | ● | ○ | ○ | ● | ○ | ● | ○ | ○ | ● |
| SmartCheck | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ |

continued on next page

Table S3 (cont'd): Characteristics of analysis tools – part 2

| Tool | Symbolic Execution | Formal Verification | Model Checking | Abstract Interpretation | Fuzzing | Runtime Verification | Concolic Testing | Mutation Testing | Pattern Matching | Code Instrumentation | Machine Learning | Taint Analysis |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SmartEmbed | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ● | ○ |
| SmartInspect | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ |
| SMARTSCOPY / SOLAR | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| SMARTSHIELD | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ |
| *SMT-based verification* | ○ | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| SODA | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ● | ○ | ○ |
| SolAnalyser | ○ | ○ | ○ | ○ | ○ | ● | ○ | ● | ○ | ● | ○ | ○ |
| SOLAR | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| SOLC-VERIFY | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Solgraph | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Solhint | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ |
| SoliAudit | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ |
| Solicitous | ○ | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| SolidiFI | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Solidifier | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| SolidiKeY | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| SolidityCheck | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ● | ○ | ○ |
| SolMet | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ |
| Solythesis | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ● | ○ | ○ |
| STAN | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Syrup | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| teEther | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| *Transaction-based analysis with LSTM network* | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ |
| Vandal | ● | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| VeriSmart | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Vertigo | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ |
| VerX | ● | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ |
| Visualgas | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| VulDeeSmartContract | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ |
| ZEUS | ● | ● | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |

## 3 QUALITY APPRAISAL

In section, we document the quality appraisal of the surveys in table S4 and the primary studies in table S5. The quality appraisal of the SLRs is contained in the main paper.

Table S4: Quality appraisal of surveys.

| survey | IDQ score | CDQ Q2 | Q3 | Q4 | Q5 | Q6 | Q7 | score | value | FDQ score | value | selected |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (Ahrendt et al., 2018) | 0.4 | reclassified as primary study | | | | | | | | | | |
| (Alkhalifah et al., 2019) | 0.4 | 0.3 | 1.0 | 0.0 | 0.0 | | 0.6 | 0.38 | low | 0.39 | low | ✗ |
| (Almakhour et al., 2020b) | 1.0 | 0.6 | 0.0 | 0.6 | 0.6 | | 1.0 | 0.56 | med | 0.78 | med | ✓ |
| (Almakhour et al., 2020a) | 0.8 | 1.0 | 0.3 | 1.0 | 0.6 | | 1.0 | 0.78 | med | 0.79 | med | ✓ |
| (Atzei et al., 2017) | 0.8 | 1.0 | 1.0 | 0.3 | 0.3 | | 0.0 | 0.52 | med | 0.66 | med | ✓ |
| (Chen et al., 2020a)• | 1.0 | 1.0 | 1.0 | 1.0 | 0.6 | | 1.0 | 0.92 | high | 0.96 | high | ✓ |
| (Chen et al., 2020b) | 1.0 | 0.3 | 1.0 | 0.3 | 0.3 | 1.0 | 1.0 | 0.65 | med | 0.83 | high | ✓ |
| (Chen et al., 2019) | 1.0 | 0.6 | 0.0 | 0.3 | 1.0 | | 0.6 | 0.50 | med | 0.75 | med | ✓ |
| (Connelly, 2020) | 0.2 | 0.3 | 1.0 | 0.0 | 1.0 | | 1.0 | 0.66 | med | 0.43 | low | ✗ |
| (Danielius et al., 2020) | 0.8 | 0.3 | 0.0 | 0.0 | 1.0 | 1.0 | 1.0 | 0.55 | med | 0.68 | med | ✓ |
| (Delmolino et al., 2016) | 0.8 | 0.3 | 1.0 | 0.0 | 0.0 | | 0.0 | 0.26 | low | 0.53 | med | ✗ |
| (Demir et al., 2019) | 0.8 | 1.0 | 1.0 | 0.3 | 0.3 | | 0.6 | 0.64 | med | 0.72 | med | ✓ |
| (Destefanis et al., 2018) | 0.4 | 0.3 | 0.0 | 0.0 | 0.0 | | 0.3 | 0.12 | low | 0.26 | low | ✗ |
| (di Angelo and Salzer, 2019a) | 0.4 | 0.3 | 0.0 | 0.0 | 0.0 | | 0.6 | 0.18 | low | 0.29 | low | ✗ |
| (di Angelo and Salzer, 2019b) | 0.4 | 1.0 | 1.0 | 1.0 | 0.6 | | 0.6 | 0.84 | high | 0.62 | med | ✓ |
| (Dika, 2017) | 0.2 | 1.0 | 1.0 | 0.6 | 1.0 | | 0.0 | 0.72 | med | 0.46 | low | ✗ |
| (Dika and Nowostawski, 2018) | 0.4 | 1.0 | 1.0 | 0.6 | 1.0 | 1.0 | 0.3 | 0.82 | high | 0.61 | med | ✓ |
| (Dingman et al., 2019a) | 0.5 | largely the same as Dingman et al. (2019b) | | | | | | | | | | |
| (Dingman et al., 2019b) | 0.5 | 0.6 | 1.0 | 0.3 | 0.0 | | 0.6 | 0.50 | med | 0.50 | med | ✓ |
| (Durieux et al., 2020) | 1.0 | 1.0 | 0.6 | 1.0 | 1.0 | 1.0 | 1.0 | 0.93 | high | 0.97 | high | ✓ |
| (Feng et al., 2019a) | 0.2 | 1.0 | 0.6 | 0.3 | 0.6 | | 0.6 | 0.62 | med | 0.41 | low | ✗ |
| (Fontein, 2018) | 0.2 | 1.0 | 0.6 | 0.0 | 1.0 | | 0.3 | 0.58 | med | 0.39 | low | ✗ |
| (Garfatta et al., 2021)⋆ | 0.4 | 0.3 | 0.3 | 0.6 | 0.6 | | 1.0 | 0.70 | med | 0.55 | med | ✓ |
| (Ghosh et al., 2020) | 1.0 | 0.3 | 0.3 | 0.0 | 0.0 | | 1.0 | 0.32 | low | 0.66 | med | ✗ |
| (Groce et al., 2020)⋆ | 0.8 | 1.0 | 1.0 | 0.0 | 0.0 | | 1.0 | 0.60 | med | 0.70 | med | ✓ |
| (Gupta and Shukla, 2019) | 0.4 | 0.3 | 0.0 | 0.0 | 0.0 | | 0.6 | 0.18 | low | 0.29 | low | ✗ |
| (Gupta, 2019) | 0.2 | largely the same as Gupta et al. (2020a) | | | | | | | | | | |
| (Gupta et al., 2020a)• | 0.8 | 1.0 | 1.0 | 0.3 | 1.0 | 1.0 | 1.0 | 0.88 | high | 0.84 | high | ✓ |
| (Gupta et al., 2020b) | 1.0 | 0.3 | 0.6 | 0.3 | 0.3 | | 1.0 | 0.50 | med | 0.75 | med | ✓ |
| (Hajdu et al., 2020) | 1.0 | reclassified as primary study | | | | | | | | | | |
| (Hartel and Schumi, 2020) | 0.8 | reclassified as primary study | | | | | | | | | | |
| (Harz and Knottenbelt, 2018) | 0.2 | 0.6 | 0.3 | 0.6 | 0.6 | | 0.3 | 0.48 | low | 0.34 | low | ✗ |
| (He et al., 2020a) | 1.0 | 1.0 | 0.6 | 0.0 | 0.3 | | 1.0 | 0.58 | med | 0.79 | med | ✓ |

⋆Identified subsequently, •Reclassified to survey from SLR

continued on next page

Table S4 (cont'd): Quality appraisal of surveys

| survey | IDQ score | CDQ Q2 | Q3 | Q4 | Q5 | Q6 | Q7 | score | value | FDQ score | value | selected |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (He et al., 2020b) | 0.8 | 0.3 | 0.0 | 0.0 | 0.0 | | 1.0 | 0.26 | low | 0.53 | med | ✗ |
| (Huang et al., 2019) | 1.0 | 0.6 | 0.6 | 1.0 | 0.6 | | 0.6 | 0.68 | med | 0.84 | high | ✓ |
| (Imeri et al., 2020) | 0.2 | 0.6 | 0.3 | 0.3 | 0.3 | | 1.0 | 0.50 | med | 0.35 | low | ✗ |
| (Kaleem et al., 2020) | 0.4 | 0.3 | 0.6 | 0.0 | 0.0 | | 1.0 | 0.38 | low | 0.39 | low | ✗ |
| (Khan and Namin, 2020) | 0.2 | 0.6 | 1.0 | 1.0 | 0.6 | | 1.0 | 0.84 | high | 0.52 | med | ✓ |
| (Khor et al., 2020) | 0.4 | 0.3 | 0.3 | 0.3 | 0.3 | | 1.0 | 0.44 | low | 0.42 | low | ✗ |
| (Kim and Lee, 2020) | 1.0 | 0.6 | 0.6 | 0.3 | 1.0 | 1.0 | 1.0 | 0.75 | med | 0.88 | high | ✓ |
| (Krupa et al., 2020) | 0.2 | 0.6 | 1.0 | 0.0 | 0.0 | | 1.0 | 0.52 | med | 0.36 | low | ✗ |
| (Lee and Choi, 2020) | 0.2 | 0.3 | 0.3 | 0.0 | 0.0 | | 1.0 | 0.32 | low | 0.26 | low | ✗ |
| (Leid et al., 2020) | 0.5 | 1.0 | 0.6 | 0.0 | 1.0 | 1.0 | 1.0 | 0.77 | med | 0.63 | med | ✓ |
| (Li et al., 2020) | 1.0 | 0.3 | 0.6 | 0.0 | 0.0 | | 1.0 | 0.38 | low | 0.69 | med | ✗ |
| (López Vivar et al., 2020) | 0.8 | 1.0 | 0.6 | 1.0 | 0.6 | | 1.0 | 0.84 | high | 0.78 | med | ✓ |
| (Lu et al., 2019) | 0.8 | reclassified as primary study | | | | | | | | | | |
| (Magazzeni et al., 2017) | 1.0 | 0.3 | 0.3 | 0.0 | 0.0 | | 0.0 | 0.12 | low | 0.56 | med | ✗ |
| (Mense and Flatscher, 2018) | 0.5 | 0.6 | 1.0 | 0.3 | 0.6 | | 0.3 | 0.56 | med | 0.53 | med | ✓ |
| (Miller et al., 2018)* | 0.5 | 0.3 | 0.3 | 0.6 | 0.6 | | 0.3 | 0.56 | med | 0.53 | med | ✓ |
| (Min and Cai, 2019) | 0.4 | 0.3 | 0.3 | 0.0 | 1.0 | 1.0 | 0.6 | 0.53 | med | 0.47 | low | ✗ |
| (Moona and Mathew, 2021) | 0.2 | 1.0 | 1.0 | 0.6 | 0.6 | | 1.0 | 0.84 | high | 0.52 | med | ✓ |
| (Murray and Anisi, 2019) | 0.4 | 0.6 | 0.0 | 0.3 | 0.6 | | 0.6 | 0.42 | low | 0.41 | low | ✗ |
| (Nguyen et al., 2019) | 0.8 | reclassified as primary study | | | | | | | | | | |
| (Pankov, 2020) | 0.4 | 0.6 | 0.3 | 0.3 | 0.3 | | 1.0 | 0.50 | med | 0.45 | low | ✗ |
| (Parizi et al., 2018) | 0.4 | 1.0 | 0.3 | 0.3 | 1.0 | 1.0 | 0.3 | 0.65 | med | 0.53 | med | ✓ |
| (Park et al., 2020) | 1.0 | reclassified as primary study | | | | | | | | | | |
| (Perez and Livshits, 2019) | 0.2 | 1.0 | 0.6 | 0.6 | 1.0 | 1.0 | 0.6 | 0.80 | high | 0.50 | med | ✓ |
| (Praitheeshan et al., 2020a) | 0.8 | 0.3 | 0.3 | 0.3 | 1.0 | 1.0 | 1.0 | 0.65 | med | 0.73 | med | ✓ |
| (Praitheeshan et al., 2020b) | 0.2 | 1.0 | 1.0 | 0.6 | 0.6 | | 1.0 | 0.84 | high | 0.52 | med | ✓ |
| (Qasse et al., 2020) | 0.4 | 0.3 | 0.0 | 0.0 | 0.0 | 1.0 | 1.0 | 0.38 | low | 0.39 | low | ✗ |
| (Radu Adrian, 2018) | 0.5 | 0.6 | 0.3 | 0.3 | 0.6 | | 0.3 | 0.42 | low | 0.46 | low | ✗ |
| (Saad et al., 2020) | 1.0 | 0.3 | 0.6 | 0.0 | 0.0 | | 1.0 | 0.38 | low | 0.69 | med | ✗ |
| (Samreen and Alalfi, 2020b) | 0.2 | 1.0 | 1.0 | 0.6 | 0.6 | | 1.0 | 0.84 | high | 0.52 | med | ✓ |
| (Sayeed et al., 2020) | 1.0 | 0.6 | 0.6 | 0.6 | 0.6 | | 1.0 | 0.68 | med | 0.84 | high | ✓ |
| (Shmatko and Olkhovskyi, 2020) | 0.2 | 0.3 | 0.0 | 0.0 | 0.0 | | 1.0 | 0.26 | low | 0.23 | low | ✗ |
| (Shrivas et al., 2020) | 0.4 | 0.3 | 0.0 | 0.0 | 0.0 | | 1.0 | 0.26 | low | 0.33 | low | ✗ |
| (Staderini and Palli, 2020) | 0.2 | 1.0 | 1.0 | 0.0 | 0.0 | | 1.0 | 0.60 | med | 0.40 | low | ✗ |
| (Staderini et al., 2020) | 0.4 | 1.0 | 1.0 | 0.0 | 0.0 | | 1.0 | 0.60 | med | 0.50 | med | ✓ |
| (Tantikul and Ngamsuriyaroj, 2020) | 0.2 | 1.0 | 1.0 | 0.0 | 1.0 | 1.0 | 1.0 | 0.83 | high | 0.52 | med | ✓ |
| (Vinayak et al., 2018) | 0.4 | 0.3 | 0.6 | 0.0 | 1.0 | | 0.3 | 0.44 | low | 0.42 | low | ✗ |

*Identified subsequently, •Reclassified to survey from SLR

continued on next page

Table S4 (cont'd): Quality appraisal of surveys

| survey | IDQ score | CDQ Q2 | Q3 | Q4 | Q5 | Q6 | Q7 | score | value | FDQ score | value | selected |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (Wohrer and Zdun, 2018) | 0.4 | 0.6 | 0.6 | 0.0 | 0.0 | | 0.3 | 0.30 | low | 0.35 | low | ✗ |
| (Xu et al., 2020) | 0.4 | 1.0 | 0.6 | 0.6 | 0.6 | | 1.0 | 0.76 | med | 0.58 | med | ✓ |
| (Ye et al., 2019a) | 1.0 | 1.0 | 0.3 | 0.6 | 1.0 | 1.0 | 0.6 | 0.75 | med | 0.88 | high | ✓ |
| (Ye et al., 2019b) | 0.5 | 0.6 | 0.3 | 0.0 | 1.0 | 1.0 | 0.6 | 0.58 | med | 0.54 | med | ✓ |
| (Zhang et al., 2020a)• | 1.0 | 1.0 | 1.0 | 0.6 | 1.0 | 1.0 | 1.0 | 0.93 | high | 0.97 | high | ✓ |
| (Zhou et al., 2020)⋆ | 1.0 | 0.6 | 0.6 | 0.6 | 0.3 | 1.0 | 1.0 | 0.68 | med | 0.84 | high | ✓ |
| (Zou et al., 2019) | 1.0 | 0.3 | 0.0 | 1.0 | 0.3 | | 0.6 | 0.44 | low | 0.72 | med | ✗ |

⋆Identified subsequently, •Reclassified to survey from SLR

Table S5: Quality appraisal of primary studies

| primary study | IDQ score | CDQ Q1 | Q2 | score | value | FDQ score | value |
|---|---|---|---|---|---|---|---|
| (Abdellatif and Brousmiche, 2018) | 0.4 | 1.0 | 1.0 | 1.00 | high | 0.70 | med |
| (Ahrendt and Bubel, 2020) | 0.5 | 0.0 | 1.0 | 0.50 | med | 0.50 | med |
| (Ahrendt et al., 2019) | 0.8 | 0.6 | 1.0 | 0.80 | high | 0.80 | high |
| (Akca et al., 2019) | 0.8 | 0.3 | 1.0 | 0.65 | med | 0.73 | med |
| (Albert et al., 2019a) | 1.0 | 0.6 | 1.0 | 0.80 | high | 0.90 | high |
| (Albert et al., 2019b) | 0.8 | 1.0 | 1.0 | 1.00 | high | 0.90 | high |
| (Albert et al., 2020a) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |
| (Albert et al., 2018) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |
| (Alt and Reitwiessner, 2018) | 0.5 | 1.0 | 1.0 | 1.00 | high | 0.75 | med |
| (Amani et al., 2018) | 0.4 | 1.0 | 1.0 | 1.00 | high | 0.70 | med |
| (Annenkov et al., 2020) | 0.4 | 0.3 | 1.0 | 0.65 | med | 0.53 | med |
| (Ashouri, 2020) | 0.8 | 0.0 | 1.0 | 0.50 | med | 0.65 | med |
| (Ashraf et al., 2020) | 1.0 | 0.0 | 1.0 | 0.50 | med | 0.75 | med |
| (Beillahi et al., 2020) | 1.0 | 0.3 | 1.0 | 0.65 | med | 0.83 | high |
| (Bhargavan et al., 2016) | 0.4 | 1.0 | 1.0 | 1.00 | high | 0.70 | med |
| (Breidenbach et al., 2019) | 1.0 | 0.0 | 1.0 | 0.50 | med | 0.75 | med |
| (Breidenbach et al., 2018) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |
| (Brent et al., 2018) | 0.2 | 1.0 | 1.0 | 1.00 | high | 0.60 | med |
| (Brent et al., 2020) | 1.0 | 0.6 | 1.0 | 0.80 | high | 0.90 | high |
| (Carver and Staron, 2020) | 1.0 | 0.0 | 1.0 | 0.50 | med | 0.75 | med |
| (Chan and Jiang, 2018) | 0.8 | 0.3 | 1.0 | 0.65 | med | 0.73 | med |
| (Chang et al., 2019) | 0.8 | 1.0 | 1.0 | 1.00 | high | 0.90 | high |
| (Chapman et al., 2019) | 0.4 | 0.3 | 1.0 | 0.65 | med | 0.53 | med |
| (Chatterjee et al., 2019) | 0.8 | 1.0 | 1.0 | 1.00 | high | 0.90 | high |
| (Chatterjee et al., 2018) | 1.0 | 1.0 | 0.0 | 0.50 | med | 0.75 | med |
| (Chen et al., 2018a) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |

⋆Identified subsequently, •Reclassified to primary study from survey

continued on next page

Table S5 (cont'd): Quality appraisal of primary studies

| primary study | IDQ score | CDQ Q1 | CDQ Q2 | CDQ score | CDQ value | FDQ score | FDQ value |
|---|---|---|---|---|---|---|---|
| (Chen et al., 2017) | 0.5 | 1.0 | 1.0 | 1.00 | high | 0.75 | med |
| (Chen et al., 2021) | 1.0 | 0.0 | 1.0 | 0.50 | med | 0.75 | med |
| (Chen et al., 2018b) | 0.5 | 0.6 | 1.0 | 0.80 | high | 0.65 | med |
| (Chen et al., 2020c) | 1.0 | 0.6 | 1.0 | 0.80 | high | 0.90 | high |
| (Chen et al., 2020d) | 1.0 | 0.6 | 1.0 | 0.80 | high | 0.90 | high |
| (Covaci et al., 2018) | 1.0 | 0.3 | 1.0 | 0.65 | med | 0.83 | high |
| (Ellul and Pace, 2018) | 0.4 | 1.0 | 1.0 | 1.00 | high | 0.70 | med |
| (Feist et al., 2019) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |
| (Feng et al., 2020) | 1.0 | 0.0 | 1.0 | 0.50 | med | 0.75 | med |
| (Ferreira Torres et al., 2020) | 0.8 | 0.3 | 1.0 | 0.65 | med | 0.73 | med |
| (Ferreira Torres et al., 2019) | 1.0 | 0.6 | 1.0 | 0.80 | high | 0.90 | high |
| (Ferreira et al., 2020b) | 1.0 | 0.0 | 1.0 | 0.50 | med | 0.75 | med |
| (Frank et al., 2020) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |
| (Fu et al., 2019) | 1.0 | 0.0 | 1.0 | 0.50 | med | 0.75 | med |
| (Gao, 2020) | 1.0 | 0.0 | 1.0 | 0.50 | med | 0.75 | med |
| (Gao et al., 2019b) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |
| (Gao et al., 2020) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |
| (Gao et al., 2019a) | 0.5 | 0.6 | 1.0 | 0.80 | high | 0.65 | med |
| (Gao et al., 2019c) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |
| (Ghaleb and Pattabiraman, 2020) | 1.0 | 0.3 | 1.0 | 0.65 | med | 0.83 | high |
| (Grech et al., 2018) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |
| (Grech et al., 2019) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |
| (Grech et al., 2020) | 1.0 | 0.0 | 1.0 | 0.50 | med | 0.75 | med |
| (Grieco et al., 2020) | 1.0 | 0.6 | 1.0 | 0.80 | high | 0.90 | high |
| (Grishchenko et al., 2018c) | 0.8 | 1.0 | 0.0 | 0.50 | med | 0.65 | med |
| (Grishchenko et al., 2018b) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |
| (Grishchenko et al., 2018a) | 0.2 | 1.0 | 1.0 | 1.00 | high | 0.60 | med |
| (Grossman et al., 2017) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |
| (Hajdu et al., 2020) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |
| (Hartel and Schumi, 2020) | 0.8 | 1.0 | 1.0 | 1.00 | high | 0.90 | high |
| (He et al., 2019) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |
| (Hegedus, 2018) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |
| (Hildenbrandt et al., 2018) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |
| (Hirai, 2017) | 0.8 | 1.0 | 0.0 | 0.50 | med | 0.65 | med |
| (Hu et al., 2021b) | 1.0 | 1.0 | 0.0 | 0.50 | med | 0.75 | med |
| (Huang et al., 2021) | 1.0 | 0.0 | 1.0 | 0.50 | med | 0.75 | med |
| (Hwang and Ryu, 2020) | 1.0 | 0.0 | 1.0 | 0.50 | med | 0.75 | med |
| (Ivanova and Khritankov, 2020) | 0.8 | 0.0 | 1.0 | 0.50 | med | 0.65 | med |
| (Jiang et al., 2018) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |
| (Jiao et al., 2020) | 1.0 | 0.6 | 1.0 | 0.80 | high | 0.90 | high |
| (Kolluri et al., 2019) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |

*Identified subsequently, •Reclassified to primary study from survey

continued on next page

Table S5 (cont'd): Quality appraisal of primary studies

| primary study | IDQ score | CDQ Q1 | CDQ Q2 | CDQ score | CDQ value | FDQ score | FDQ value |
|---|---|---|---|---|---|---|---|
| (Krupp and Rossow, 2018) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |
| (Li, 2019) | 1.0 | 0.0 | 1.0 | 0.50 | med | 0.75 | med |
| (Li et al., 2020) | 1.0 | 0.6 | 1.0 | 0.80 | high | 0.90 | high |
| (Li et al., 2020) | 0.8 | 0.0 | 1.0 | 0.50 | med | 0.65 | med |
| (Li et al., 2019) | 1.0 | 0.3 | 1.0 | 0.65 | med | 0.83 | high |
| (Liao et al., 2019) | 0.4 | 0.3 | 1.0 | 0.65 | med | 0.53 | med |
| (Liu et al., 2020a) | 1.0 | 0.0 | 1.0 | 0.50 | med | 0.75 | med |
| (Liu et al., 2019) | 0.5 | 0.6 | 1.0 | 0.80 | high | 0.65 | med |
| (Liu et al., 2018b) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |
| (Liu et al., 2020b) | 1.0 | 0.0 | 1.0 | 0.50 | med | 0.75 | med |
| (Liu et al., 2018a) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |
| (Livshits, 2020) | 1.0 | 0.0 | 1.0 | 0.50 | med | 0.75 | med |
| (Lu et al., 2019) | 0.8 | 1.0 | 1.0 | 1.00 | high | 0.90 | high |
| (Luu et al., 2016) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |
| (Ma et al., 2019) | 0.5 | 1.0 | 1.0 | 1.00 | high | 0.75 | med |
| (Marescotti et al., 2020) | 0.5 | 0.0 | 1.0 | 0.50 | med | 0.50 | med |
| (Mavridou and Laszka, 2018a) | 0.8 | 1.0 | 1.0 | 1.00 | high | 0.90 | high |
| (Mavridou and Laszka, 2018b) | 0.8 | 1.0 | 1.0 | 1.00 | high | 0.90 | high |
| (Mavridou et al., 2019) | 0.8 | 1.0 | 1.0 | 1.00 | high | 0.90 | high |
| (Mei et al., 2019) | 0.8 | 0.0 | 1.0 | 0.50 | med | 0.65 | med |
| (Mossberg et al., 2019) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |
| (Nehaï and Bobot, 2019) | 1.0 | 0.0 | 1.0 | 0.50 | med | 0.75 | med |
| (Nehaï et al., 2018) | 0.4 | 1.0 | 1.0 | 1.00 | high | 0.70 | med |
| (Nelaturu et al., 2020) | 0.4 | 0.3 | 1.0 | 0.65 | med | 0.53 | med |
| (Nguyen et al., 2020) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |
| (Nguyen et al., 2019) | 0.8 | 0.0 | 1.0 | 0.50 | med | 0.65 | med |
| (Nikolić et al., 2018) | | 1.0 | 1.0 | 1.00 | high | 0.50 | med |
| (Norvill et al., 2018) | 0.8 | 0.6 | 1.0 | 0.80 | high | 0.80 | high |
| (Osterland and Rose, 2020) | 1.0 | 0.3 | 1.0 | 0.65 | med | 0.83 | high |
| (Park et al., 2018) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |
| (Peng et al., 2019) | 0.8 | 0.3 | 1.0 | 0.65 | med | 0.73 | med |
| (Permenev et al., 2020) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |
| (Pérez et al., 2020) | 1.0 | 0.0 | 1.0 | 0.50 | med | 0.75 | med |
| (Qian et al., 2020) | 1.0 | 0.3 | 1.0 | 0.65 | med | 0.83 | high |
| (Quan et al., 2019) | 0.2 | 0.6 | 1.0 | 0.80 | high | 0.50 | med |
| (Samreen and Alalfi, 2020a) | 0.4 | 0.3 | 1.0 | 0.65 | med | 0.53 | med |
| (Schneidewind et al., 2020b) | 0.5 | 0.0 | 1.0 | 0.50 | med | 0.50 | med |
| (Schneidewind et al., 2020a) | 1.0 | 0.6 | 1.0 | 0.80 | high | 0.90 | high |
| (Shishkin, 2019) | 0.5 | 0.3 | 1.0 | 0.65 | med | 0.58 | med |
| (Signer, 2018) | 0.2 | 0.6 | 1.0 | 0.80 | high | 0.50 | med |
| (So et al., 2020) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |

⋆Identified subsequently, •Reclassified to primary study from survey

continued on next page

Table S5 (cont'd): Quality appraisal of primary studies

| | IDQ | CDQ | | | | FDQ | |
|---|---|---|---|---|---|---|---|
| primary study | score | Q1 | Q2 | score | value | score | value |
| (Tan et al., 2020) | 1.0 | 0.0 | 1.0 | 0.50 | med | 0.75 | med |
| (Tann et al., 2018) | 0.2 | 0.6 | 1.0 | 1.00 | high | 0.60 | med |
| (Tikhomirov et al., 2018) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |
| (Torres et al., 2019) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |
| (Torres et al., 2018) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |
| (Tsankov, 2018) | 0.5 | 0.3 | 1.0 | 0.65 | med | 0.58 | med |
| (Tsankov et al., 2018) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |
| (Vandenbogaerde, 2019) | 1.0 | 0.0 | 1.0 | 0.50 | med | 0.75 | med |
| (Wang et al., 2019a) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |
| (Wang et al., 2020a) | 1.0 | 0.3 | 1.0 | 0.65 | med | 0.83 | high |
| (Wang et al., 2018) | 0.2 | 1.0 | 1.0 | 1.00 | high | 0.60 | med |
| (Wang et al., 2020c) | 0.8 | 0.3 | 1.0 | 0.65 | med | 0.73 | med |
| (Wang et al., 2019c) | 1.0 | 0.6 | 1.0 | 0.80 | high | 0.90 | high |
| (Wang et al., 2019b) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |
| (Wang et al., 2020b) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |
| (Wang et al., 2020d) | 1.0 | 0.3 | 1.0 | 0.65 | med | 0.83 | high |
| (Weiss and Schütte, 2019) | 1.0 | 0.3 | 1.0 | 0.65 | med | 0.83 | high |
| (Wu et al., 2020) | 0.2 | 1.0 | 1.0 | 1.00 | high | 0.60 | med |
| (Wüstholz and Christakis, 2020a) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |
| (Wüstholz and Christakis, 2020b) | 1.0 | 0.6 | 1.0 | 0.80 | high | 0.90 | high |
| (Xue et al., 2020) | 1.0 | 0.0 | 1.0 | 0.50 | med | 0.75 | med |
| (Yamashita et al., 2019) | 0.4 | 1.0 | 1.0 | 1.00 | high | 0.70 | med |
| (Yang and Lei, 2019a) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |
| (Yang and Lei, 2018b) | 1.0 | 0.3 | 1.0 | 0.65 | med | 0.83 | high |
| (Yang and Lei, 2019b) | 0.5 | 0.0 | 1.0 | 0.50 | med | 0.50 | med |
| (Yang and Lei, 2018a) | 0.5 | 0.6 | 1.0 | 0.80 | high | 0.65 | med |
| (Yang et al., 2020c) | 1.0 | 0.0 | 1.0 | 0.50 | med | 0.75 | med |
| (Yang et al., 2020a) | 0.8 | 0.0 | 1.0 | 0.50 | med | 0.65 | med |
| (Yang et al., 2020b) | 1.0 | 0.6 | 1.0 | 0.80 | high | 0.90 | high |
| (Ye et al., 2020) | 1.0 | 0.0 | 1.0 | 0.50 | med | 0.75 | med |
| (Yu et al., 2020) | 0.8 | 0.3 | 1.0 | 0.65 | med | 0.73 | med |
| (Zakrzewski, 2018) | 0.8 | 1.0 | 0.0 | 0.50 | med | 0.65 | med |
| (Zhang et al., 2016) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |
| (Zhang et al., 2020b) | 1.0 | 0.0 | 1.0 | 0.50 | med | 0.75 | med |
| (Zhang et al., 2020d) | 0.5 | 0.6 | 1.0 | 0.80 | high | 0.65 | med |
| (Zhang et al., 2020c) | 0.5 | 0.3 | 1.0 | 0.65 | med | 0.58 | med |
| (Zhang et al., 2019b) | 1.0 | 0.0 | 1.0 | 0.50 | med | 0.75 | med |
| (Zhou et al., 2018b) | 1.0 | 1.0 | 1.0 | 1.00 | high | 1.00 | high |
| (Zhou et al., 2018a) | 0.4 | 1.0 | 1.0 | 1.00 | high | 0.70 | med |
| (Zhuang et al., 2020) | 1.0 | 0.3 | 1.0 | 0.65 | med | 0.83 | high |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| (Zupan et al., 2020) | 0.4 | 0.3 | 1.0 | 0.65 | med | 0.53 | med |

*Identified subsequently, •Reclassified to primary study from survey

## REFERENCES


Abdellatif, T. and Brousmiche, K.-L. (2018). Formal verification of smart contracts based on users and blockchain behaviors models. In *9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)* (IEEE), NTMS 2018, 1–5. doi:10.1109/ntms.2018.8328737

Ahrendt, W. and Bubel, R. (2020). Functional verification of smart contracts via strong data integrity. In *Leveraging Applications of Formal Methods, Verification and Validation: Applications* (Springer International Publishing), vol. 12478 of *ISoLA 2020, Lecture Notes in Computer Science (LNCS)*, 9–24. doi:10.1007/978-3-030-61467-6_2

Ahrendt, W., Bubel, R., Ellul, J., Pace, G. J., Pardo, R., Rebiscoul, V., et al. (2019). Verification of smart contract business logic. In *International Conference on Fundamentals of Software Engineering* (Springer International Publishing), vol. 11761 of *FSEN 2019, Lecture Notes in Computer Science (LNCS)*, 228–243. doi:10.1007/978-3-030-31517-7_16

Ahrendt, W., Pace, G. J., and Schneider, G. (2018). Smart contracts: A killer application for deductive source code verification. In *Principled Software Development: Essays Dedicated to Arnd Poetzsch-Heffter on the Occasion of his 60th Birthday* (Springer International Publishing). 1–18. doi:10.1007/978-3-319-98047-8_1

Akca, S., Rajan, A., and Peng, C. (2019). Solanalyser: A framework for analysing and testing smart contracts. In *26th Asia-Pacific Software Engineering Conference (APSEC)* (IEEE), APSEC 19, 482–489. doi:10.1109/APSEC48747.2019.00071

Albert, E., Correas, J., Gordillo, P., Román-Díez, G., and Rubio, A. (2019a). Safevm: A safety verifier for ethereum smart contracts. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis* (New York, NY, USA: Association for Computing Machinery), ISSTA 2019, 386–389. doi:10.1145/3293882.3338999

Albert, E., Correas, J., Gordillo, P., Román-Díez, G., and Rubio, A. (2020a). Gasol: Gas analysis and optimization for ethereum smart contracts. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (Springer International Publishing), vol. 12079 of *TACAS 2020, Lecture Notes in Computer Science (LNCS)*, 118–125. doi:10.1007/978-3-030-45237-7_7

Albert, E., Gordillo, P., Livshits, B., Rubio, A., and Sergey, I. (2018). Ethir: A framework for high-level analysis of ethereum bytecode. In *International Symposium on Automated Technology for Verification and Analysis*, eds. S. K. Lahiri and C. Wang (Cham: Springer International Publishing), vol. 11138 of *ATVA 2018*, 513–520. doi:10.1007/978-3-030-01090-4_30

Albert, E., Gordillo, P., Rubio, A., and Schett, M. A. (2020b). Synthesis of super-optimized smart contracts using max-smt. In *Computer Aided Verification*, eds. S. K. Lahiri and C. Wang (Springer International Publishing), 177–200. doi:10.1007/978-3-030-53288-8_10

Albert, E., Gordillo, P., Rubio, A., and Sergey, I. (2019b). Running on fumes: Preventing out-of-gas vulnerabilities in ethereum smart contracts using static resource analysis. In *International Conference on Verification and Evaluation of Computer and Communication Systems* (Springer International Publishing), vol. 11847 of *VECoS2019, Lecture Notes in Computer Science (LNCS)*, 63–78. doi:10.1007/978-3-030-35092-5_5

Alkhalifah, A., Ng, A., Chowdhury, M. J. M., Kayes, A. S. M., and Watters, P. A. (2019). An empirical analysis of blockchain cybersecurity incidents. In *IEEE Asia-Pacific Conference on Computer Science*

*and Data Engineering (CSDE)* (IEEE), 1–8. doi:10.1109/CSDE48274.2019.9162381

Almakhour, M., Sliman, L., Samhat, A. E., and Mellouk, A. (2020a). On the verification of smart contracts: A systematic review. In *International Conference on Blockchain* (Springer International Publishing), ICBC 2020, Lecture Notes in Computer Science (LNCS), 94–107. doi:10.1007/978-3-030-59638-5_7

Almakhour, M., Sliman, L., Samhat, A. E., and Mellouk, A. (2020b). Verification of smart contracts: A survey. *Pervasive and Mobile Computing* 67, 101227. doi:10.1016/j.pmcj.2020.101227

Alt, L. and Reitwiessner, C. (2018). Smt-based verification of solidity smart contracts. In *International Symposium on Leveraging Applications of Formal Methods* (Springer International Publishing), vol. 11247 of *ISoLA 2018, Lecture Notes in Computer Science (LNCS)*, 376–388. doi:10.1007/978-3-030-03427-6_28

Amani, S., Begel, M., Bortin, M., and Staples, M. (2018). Towards verifying ethereum smart contract bytecode in isabelle/hol. In *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs* (New York, NY, USA: Association for Computing Machinery), CPP 2018, 66–77. doi:10.1145/3167084

Annenkov, D., Nielsen, J. B., and Spitters, B. (2020). Concert: A smart contract certification framework in coq. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs* (New York, NY, USA: Association for Computing Machinery), CPP 2020, 215–228. doi:10.1145/3372885.3373829

Antonino, P. and Roscoe, A. W. (2020). Formalising and verifying smart contracts with solidifier: a bounded model checker for solidity. *arXiv preprint arXiv:2002.02710*

Antonio Pierro, G. and Tonelli, R. (2020). Paso: A web-based parser for solidity language analysis. In *IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE)* (IEEE), 16–21. doi:10.1109/IWBOSE50093.2020.9050263

Ashouri, M. (2020). Etherolic: A practical security analyzer for smart contracts. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing* (Association for Computing Machinery), SAC 20, 353–356. doi:10.1145/3341105.3374226

Ashraf, I., Ma, X., Jiang, B., and Chan, W. K. (2020). Gasfuzzer: Fuzzing ethereum smart contract binaries to expose gas-oriented exception security vulnerabilities. *IEEE Access* 8, 99552–99564. doi:10.1109/ACCESS.2020.2995183

Atzei, N., Bartoletti, M., and Cimoli, T. (2017). A survey of attacks on ethereum smart contracts (sok). In *International Conference on Principles of Security and Trust* (Springer Berlin Heidelberg), vol. 10204 of *POST 2017, Lecture Notes in Computer Science (LNCS)*, 164–186. doi:10.1007/978-3-662-54455-6_8

Bansal, K., Koskinen, E., and Tripp, O. (2018). Automatic generation of precise and useful commutativity conditions. In *Tools and Algorithms for the Construction and Analysis of Systems*, eds. D. Beyer and M. Huisman (Springer International Publishing), vol. 10805 of *LNCS*, 115–132. doi:10.1007/978-3-319-89960-2_7

Beillahi, S. M., Ciocarlie, G., Emmi, M., and Enea, C. (2020). Behavioral simulation for smart contracts. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation* (New York, NY, USA: Association for Computing Machinery), PLDI 2020, 470–486. doi:10.1145/3385412.3386022

Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Gollamudi, A., Gonthier, G., Kobeissi, N., et al. (2016). Formal verification of smart contracts: Short paper. In *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security* (Association for Computing Machinery), PLAS 16, 91–96. doi:10.1145/2993600.2993611

Bragagnolo, S., Rocha, H., Denker, M., and Ducasse, S. (2018). Smartinspect: solidity smart contract inspector. In *International Workshop on Blockchain Oriented Software Engineering (IWBOSE)* (IEEE),

9–18. doi:10.1109/IWBOSE.2018.8327566

Breidenbach, L., Daian, P., Tramer, F., and Juels, A. (2018). Enter the hydra: Towards principled bug bounties and exploit-resistant smart contracts. In *Proceedings of the 27th USENIX Conference on Security Symposium* (USENIX Association), USENIX Security 2018, 1335–1352

Breidenbach, L., Daian, P., Tramer, F., and Juels, A. (2019). The hydra framework for principled, automated bug bounties. *IEEE Security Privacy* 17, 53–61. doi:10.1109/MSEC.2019.2914109

Brent, L., Grech, N., Lagouvardos, S., Scholz, B., and Smaragdakis, Y. (2020). Ethainter: A smart contract security analyzer for composite vulnerabilities. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation* (Association for Computing Machinery), PLDI 2020, 454–469. doi:10.1145/3385412.3385990

Brent, L., Jurisevic, A., Kong, M., Liu, E., Gauthier, F., Gramoli, V., et al. (2018). Vandal: A scalable security analysis framework for smart contracts. *arXiv preprint arXiv:1809.03981*

Carver, J. C. and Staron, M. (2020). Blockchain and smart contract engineering. *IEEE Software* 37, 94–96. doi:10.1109/MS.2020.2999995

Chan, W. K. and Jiang, B. (2018). Fuse: An architecture for smart contract fuzz testing service. In *25th Asia-Pacific Software Engineering Conference (APSEC)* (IEEE), APSEC 18, 707–708. doi:10.1109/APSEC.2018.00099

Chang, J., Gao, B., Xiao, H., Sun, J., Cai, Y., and Yang, Z. (2019). scompile: Critical path identification and analysis for smart contracts. In *Formal Methods and Software Engineering*, eds. Y. Ait-Ameur and S. Qin (Springer International Publishing), vol. 11852 of *ICFEM 2019*, 286–304. doi:10.1007/978-3-030-32409-4_18

Chapman, P., Xu, D., Deng, L., and Xiong, Y. (2019). Deviant: a mutation testing tool for solidity smart contracts. In *IEEE International Conference on Blockchain (Blockchain)* (IEEE), 319–324. doi:10.1109/blockchain.2019.00050

Chatterjee, K., Goharshady, A. K., and Goharshady, E. K. (2019). The treewidth of smart contracts. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing* (New York, NY, USA: Association for Computing Machinery), ACM 2019, 400–408. doi:10.1145/3297280.3297322

Chatterjee, K., Goharshady, A. K., and Velner, Y. (2018). Quantitative analysis of smart contracts. In *27th European Symposium on Programming: Programming Languages and Systems* (Springer International Publishing), vol. 10801 of *ESOP 2018*, 739–767. doi:10.1007/978-3-319-89884-1_26

Chen, H., Pendleton, M., Njilla, L., and Xu, S. (2020a). A survey on ethereum systems security: Vulnerabilities, attacks, and defenses. *ACM Computing Surveys* 53, 1–43. doi:10.1145/3391195

Chen, J., Xia, X., Lo, D., Grundy, J., Luo, X., and Chen, T. (2020b). Defining smart contract defects on ethereum. *IEEE Transactions on Software Engineering* , 1–1doi:10.1109/TSE.2020.2989002

Chen, J., Xia, X., Lo, D., Grundy, J., Luo, X., and Chen, T. (2021). DEFECTCHECKER: Automated smart contract defect detection by analyzing evm bytecode. *IEEE Transactions on Software Engineering* doi:10.1109/tse.2021.3054928

Chen, T., Cao, R., Li, T., Luo, X., Gu, G., Zhang, Y., et al. (2020c). Soda: A generic online detection framework for smart contracts. In *27th Annual Network and Distributed System Security Symposium* (The Internet Society), NDSS 2020, 1–17. doi:10.14722/ndss.2020.24449

Chen, T., Feng, Y., Li, Z., Zhou, H., Luo, X., Li, X., et al. (2020d). Gaschecker: Scalable analysis for discovering gas-inefficient smart contracts. *IEEE Transactions on Emerging Topics in Computing* doi:10.1109/tetc.2020.2979019

Chen, T., Li, X., Luo, X., and Zhang, X. (2017). Under-optimized smart contracts devour your money. In *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering* (IEEE),

SANER 17, European Conference on Software Maintenance and Reengineering (CSMR), 442–446. doi:10.1109/SANER.2017.7884650

Chen, T., Li, Z., Zhang, Y., Luo, X., Wang, T., Hu, T., et al. (2019). A large-scale empirical study on control flow identification of smart contracts. In *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)* (IEEE), ESEM 19, 1–11. doi:10.1109/ESEM.2019.8870156

Chen, T., Li, Z., Zhou, H., Chen, J., Luo, X., Li, X., et al. (2018a). Towards saving money in using smart contracts. In *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results* (New York, NY, USA: Association for Computing Machinery), ICSE (NIER) 2018, 81–84. doi:10.1145/3183399.3183420

Chen, X., Park, D., and Roşu, G. (2018b). A language-independent approach to smart contract verification. In *International Symposium on Leveraging Applications of Formal Methods*, eds. T. Margaria and B. Steffen (Springer International Publishing), vol. 11247 of *ISoLA 2018, Lecture Notes in Computer Science (LNCS)*, 405–413. doi:10.1007/978-3-030-03427-6_30

Chinen, Y., Yanai, N., Cruz, J. P., and Okamura, S. (2020). Ra: Hunting for re-entrancy attacks in ethereum smart contracts via static analysis. In *IEEE International Conference on Blockchain (Blockchain)* (IEEE), 327–336. doi:10.1109/blockchain50366.2020.00048

Connelly, D. S. (2020). *Smart Contract Vulnerabilities on the Ethereum Blockchain: A Current Perspective*. Master's thesis, Portland State University. doi:10.15760/etd.7313

[Dataset] ContractGuard (2018). Contractguard. `https://guard-strike.oss-cn-shanghai.aliyuncs.com/contract-guard/ContractGuard-WhitePaper-V1.0.pdf`. Accessed: 2021-08-07

Cook, T., Latham, A., and Lee, J. H. (2017). *DappGuard: Active Monitoring and Defense for Solidity Smart Contracts*. Tech. rep., Massachusetts Institute of Technology (MIT)

Covaci, A., Madeo, S., Motylinski, P., and Vincent, S. (2018). Nectar: Non-interactive smart contract protocol using blockchain technology. In *1st IEEE/ACM International Workshop on Emerging Trends in Software Engineering for Blockchain, WETSEB@ICSE 2018* (New York, NY, USA: Association for Computing Machinery), WETSEB 18, 17–24. doi:10.1145/3194113.3194116

Danielius, P., Stolarski, P., and Masteika, S. (2020). Vulnerabilities and excess gas consumption analysis within ethereum-based smart contracts for electricity market. In *International Conference on Business Information Systems*, eds. W. Abramowicz and G. Klein (Springer International Publishing), BIS 2020, Lecture Notes in Business Information Processing (LNBIP), 99–110. doi:10.1007/978-3-030-61146-0_8

Delmolino, K., Arnett, M., Kosba, A., Miller, A., and Shi, E. (2016). Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab. In *Financial Cryptography and Data Security* (Springer Berlin Heidelberg), vol. 9604 of *Lecture Notes in Computer Science LNCS*, 79–94. doi:10.1007/978-3-662-53357-4_6

Demir, M., Alalfi, M., Turetken, O., and Ferworn, A. (2019). Security smells in smart contracts. In *IEEE 19th International Conference on Software Quality, Reliability and Security Companion* (IEEE), QRS 19, 442–449. doi:10.1109/qrs-c.2019.00086

Destefanis, G., Marchesi, M., Ortu, M., Tonelli, R., Bracciali, A., and Hierons, R. (2018). Smart contracts vulnerabilities: a call for blockchain software engineering? In *International Workshop on Blockchain Oriented Software Engineering (IWBOSE)* (IEEE), 19–25. doi:10.1109/iwbose.2018.8327567

di Angelo, M. and Salzer, G. (2019a). Collateral use of deployment code for smart contracts in ethereum. In *10th IFIP International Conference on New Technologies, Mobility and Security (NTMS)* (IEEE), IFIP NTMS 19, 1–5. doi:10.1109/NTMS.2019.8763828

di Angelo, M. and Salzer, G. (2019b). A survey of tools for analyzing ethereum smart contracts. In *IEEE International Conference on Decentralized Applications and Infrastructures (DAPPCON)* (IEEE), DAPPCON 2019, 69–78. doi:10.1109/DAPPCON.2019.00018

Dika, A. (2017). *Ethereum Smart Contracts: Security Vulnerabilities and Security Tools*. Master's thesis, Norwegian University of Science and Technology, Department of Computer Science

Dika, A. and Nowostawski, M. (2018). Security vulnerabilities in ethereum smart contracts. In *IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)* (IEEE), iThings/GreenCom/CPSCom/SmartData 2018, 955–962. doi:10.1109/cybermatics_2018.2018.00182

Dingman, W., Cohen, A., Ferrara, N., Lynch, A., Jasinski, P., Black, P. E., et al. (2019a). Classification of smart contract bugs using the nist bugs framework. In *IEEE 17th International Conference on Software Engineering Research, Management and Applications (SERA)* (IEEE), SERA 19, 116–123. doi:10.1109/sera.2019.8886793

Dingman, W., Cohen, A., Ferrara, N., Lynch, A., Jasinski, P., Black, P. E., et al. (2019b). Defects and vulnerabilities in smart contracts, a classification using the nist bugs framework. *International Journal of Networked and Distributed Computing* 7, 121–132. doi:10.2991/ijndc.k.190710.003

Ducasse, S., Rocha, H., Bragagnolo, S., Denker, M., and Francomme, C. (2019). SmartAnvil: Open-source tool suite for smart contractanalysis. In *Blockchain and Web 3.0: Social, Economic, and Technological Challenges* (Routledge). 31 pages. doi:10.4324/9780429029530-13

Durieux, T., Ferreira, J. F., Abreu, R., and Cruz, P. (2020). Empirical review of automated analysis tools on 47,587 ethereum smart contracts. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering* (Association for Computing Machinery), ICSE 20, 530–541. doi:10.1145/3377811.3380364

Ellul, J. and Pace, G. J. (2018). Runtime verification of ethereum smart contracts. In *14th European Dependable Computing Conference (EDCC)* (IEEE), EDCC 18, 158–163. doi:10.1109/edcc.2018.00036

Feist, J., Greico, G., and Groce, A. (2019). Slither: A static analysis framework for smart contracts. In *Proceedings of the 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain, WETSEB@ICSE 2019* (IEEE), WETSEB@ICSE 2019, 8–15. doi:10.1109/wetseb.2019.00008

Feng, X., Wang, Q., Zhu, X., and Wen, S. (2019a). Bug searching in smart contract. *arXiv preprint arXiv:1905.00799*

Feng, Y., Torlak, E., and Bodik, R. (2019b). Precise attack synthesis for smart contracts. *arXiv preprint arXiv:1902.06067*

Feng, Y., Torlak, E., and Bodik, R. (2020). Summary-based symbolic evaluation for smart contracts. In *35th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (ACM), ASE 20, 1141–1152. doi:10.1145/3324884.3416646

[Dataset] Ferreira, J. F., Cruz, P., Durieux, T., and Abreu, R. (2020a). Smartbugs. `https://github.com/smartbugs/smartbugs`. Accessed: 2021-08-04

Ferreira, J. F., Cruz, P., Durieux, T., and Abreu, R. (2020b). Smartbugs: A framework to analyze solidity smart contracts. In *35th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (ACM), ASE 20, 1349–1352. doi:10.1145/3324884.3415298

Ferreira Torres, C., Baden, M., Norvill, R., Fiz Pontiveros, B. B., Jonker, H., and Mauw, S. (2020). Aegis: Shielding vulnerable smart contracts against attacks. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security* (New York, NY, USA: Association for Computing Machinery),

ASIA CCS 2020, 584–597. doi:10.1145/3320269.3384756

Ferreira Torres, C., Baden, M., Norvill, R., and Jonker, H. (2019). Aegis: Smart shielding of smart contracts. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA: Association for Computing Machinery), CCS 19, 2589–2591. doi:10.1145/3319535.3363263

Fontein, R. (2018). *Comparison of static analysis tooling for smart contracts on the evm*. Bachelor's thesis, University of Twente, Twente Student conference on IT

Frank, J., Aschermann, C., and Holz, T. (2020). ETHBMC: A bounded model checker for smart contracts. In *29th USENIX Security Symposium (USENIX Security 20)* (USENIX Association), USENIX Security 2020, 2757–2774

Fu, M., Wu, L., Hong, Z., Zhu, F., Sun, H., and Feng, W. (2019). A critical-path-coverage-based vulnerability detection method for smart contracts. *IEEE Access* 7, 147327–147344. doi:10.1109/ACCESS.2019.2947146

Gao, J., Liu, H., Li, Y., Liu, C., Yang, Z., Li, Q., et al. (2019a). Towards automated testing of blockchain-based decentralized applications. In *IEEE/ACM 27th International Conference on Program Comprehension (ICPC)* (IEEE), 294–299. doi:10.1109/ICPC.2019.00048

Gao, J., Liu, H., Liu, C., Li, Q., Guan, Z., and Chen, Z. (2019b). Easyflow: Keep ethereum away from overflow. In *IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)* (IEEE), ISCE 19, 23–26. doi:10.1109/ICSE-Companion.2019.00029

Gao, Z. (2020). When deep learning meets smart contracts. In *35th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (ACM), ASE 20, 1400–1402. doi:10.1145/3324884.3418918

Gao, Z., Jayasundara, V., Jiang, L., Xia, X., Lo, D., and Grundy, J. (2019c). Smartembed: A tool for clone and bug detection in smart contracts through structural code embedding. In *IEEE International Conference on Software Maintenance and Evolution (ICSME)* (IEEE), ICSME 2019, 394–397. doi:10.1109/icsme.2019.00067

Gao, Z., Jiang, L., Xia, X., Lo, D., and Grundy, J. (2020). Checking smart contracts with structural code embedding. *IEEE Transactions on Software Engineering* , 1–1doi:10.1109/TSE.2020.2971482

Garfatta, I., Klai, K., Gaaloul, W., and Graiet, M. (2021). A survey on formal verification for solidity smart contracts. In *Australasian Computer Science Week Multiconference* (Association for Computing Machinery), ACSW 21, 1–10. doi:10.1145/3437378.3437879

Ghaleb, A. and Pattabiraman, K. (2020). How effective are smart contract analysis tools? evaluating smart contract static analysis tools using bug injection. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis* (Association for Computing Machinery), ISSTA 2020, 415–427. doi:10.1145/3395363.3397385

Ghosh, A., Gupta, S., Dua, A., and Kumar, N. (2020). Security of cryptocurrencies in blockchain technology: State-of-art, challenges and future prospects. *Journal of Network and Computer Applications* 163, 102635. doi:10.1016/j.jnca.2020.102635

Grech, N., Brent, L., Scholz, B., and Smaragdakis, Y. (2019). Gigahorse: Thorough, declarative decompilation of smart contracts. In *Proceedings of the 41st International Conference on Software Engineering* (IEEE), ICSE 19, 1176–1186. doi:10.1109/ICSE.2019.00120

Grech, N., Kong, M., Jurisevic, A., Brent, L., Scholz, B., and Smaragdakis, Y. (2018). Madmax: Surviving out-of-gas conditions in ethereum smart contracts. *Proceedings of the ACM on Programming Languages (PACMPL)* 2, 27. doi:10.1145/3276486

Grech, N., Kong, M., Jurisevic, A., Brent, L., Scholz, B., and Smaragdakis, Y. (2020). Madmax: Analyzing the out-of-gas world of smart contracts. *Communications of the ACM* 63, 87–95. doi:10.1145/3416262

Grieco, G., Song, W., Cygan, A., Feist, J., and Groce, A. (2020). Echidna: Effective, usable, and fast fuzzing for smart contracts. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis* (New York, NY, USA: Association for Computing Machinery), ISSTA 2020, 557–560. doi:10.1145/3395363.3404366

Grishchenko, I., Maffei, M., and Schneidewind, C. (2018a). *EtherTrust: sound static analysis of Ethereum bytecode*. Tech. rep., Technische Universität Wien

Grishchenko, I., Maffei, M., and Schneidewind, C. (2018b). Foundations and tools for the static analysis of ethereum smart contracts. In *International Conference on Computer Aided Verification*, eds. H. Chockler and G. Weissenbacher (Springer International Publishing), vol. 10981 of *CAV 2018, Lecture Notes in Computer Science (LNCS)*, 51–78. doi:10.1007/978-3-319-96145-3_4

Grishchenko, I., Maffei, M., and Schneidewind, C. (2018c). A semantic framework for the security analysis of ethereum smart contracts. In *International Conference on Principles of Security and Trust*, eds. L. Bauer and R. Küsters (Springer International Publishing), vol. 10804 of *POST 2018, Lecture Notes in Computer Science (LNCS)*, 243–269. doi:10.1007/978-3-319-89722-6_10

Groce, A., Feist, J., Grieco, G., and Colburn, M. (2020). What are the actual flaws in important smart contracts (and how can we find them)? In *Financial Cryptography and Data Security*, eds. J. Bonneau and N. Heninger (Springer International Publishing), vol. 12059 of *FC 2020*, 634–653. doi:10.1007/978-3-030-51280-4_34

Groce, A. and Grieco, G. (2021). Echidna-parade: A tool for diverse multicore smart contract fuzzing. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis* (Association for Computing Machinery), ISSTA 2021, 658–661. doi:10.1145/3460319.3469076

Grossman, S., Abraham, I., Golan-Gueta, G., Michalevsky, Y., Rinetzky, N., Sagiv, M., et al. (2017). Online detection of effectively callback free objects with applications to smart contracts. *Proceedings of the ACM on Programming Languages (PACMPL)* 2, 28. doi:10.1145/3158136

Gupta, B. C. (2019). *Analysis of Ethereum Smart Contracts - A Security Perspective*. Master's thesis, Department of Computer Science and Engineering, Indian Institute of Technology Kanpur

Gupta, B. C., Kumar, N., Handa, A., and Shukla, S. K. (2020a). An insecurity study of ethereum smart contracts. In *International Conference on Security, Privacy, and Applied Cryptography Engineering (SPACE)* (Springer International Publishing), vol. 12586 of *SPACE 2020, Lecture Notes in Computer Science (LNCS)*, 188–207. doi:10.1007/978-3-030-66626-2_10

Gupta, B. C. and Shukla, S. K. (2019). A study of inequality in the ethereum smart contract ecosystem. In *Sixth International Conference on Internet of Things: Systems, Management and Security* (IEEE), IoTSMS 2019, 441–449. doi:10.1109/IOTSMS48152.2019.8939257

Gupta, R., Tanwar, S., Al-Turjman, F., Italiya, P., Nauman, A., and Kim, S. W. (2020b). Smart contract privacy protection using ai in cyber-physical systems: Tools, techniques and challenges. *IEEE Access* 8, 24746–24772. doi:10.1109/ACCESS.2020.2970576

Hajdu, A., Ivaki, N., Kocsis, I., Klenik, A., Gonczy, L., Laranjeiro, N., et al. (2020). Using fault injection to assess blockchain systems in presence of faulty smart contracts. *IEEE Access* 8, 190760–190783. doi:10.1109/ACCESS.2020.3032239

Hajdu, Á. and Jovanović, D. (2020). solc-verify: A modular verifier for solidity smart contracts. In *Verified Software. Theories, Tools, and Experiments*, eds. S. Chakraborty and J. A. Navas (Springer International Publishing), 161–179. doi:10.1007/978-3-030-41600-3_11

Hartel, P. and Schumi, R. (2020). Mutation testing of smart contracts at scale. In *International Conference on Tests and Proofs*, eds. W. Ahrendt and H. Wehrheim (Springer International Publishing), TAP 2020, Lecture Notes in Computer Science (LNCS), 23–42. doi:10.1007/978-3-030-50995-8_2

Hartel, P. and van Staalduinen, M. (2019). Truffle tests for free – replaying ethereum smart contracts for transparency. *arXiv preprint arXiv:1907.09208*

Harz, D. and Knottenbelt, W. (2018). Towards safer smart contracts: A survey of languages and verification methods. *arXiv preprint arXiv:1809.09805*

He, D., Deng, Z., Zhang, Y., Chan, S., Cheng, Y., and Guizani, N. (2020a). Smart contract vulnerability analysis and security audit. *IEEE Network* 34, 276–282. doi:10.1109/mnet.001.1900656

He, J., Balunović, M., Ambroladze, N., Tsankov, P., and Vechev, M. (2019). Learning to fuzz from symbolic execution with application to smart contracts. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA: Association for Computing Machinery), CCS 19, 531–548. doi:10.1145/3319535.3363230

He, N., Wu, L., Wang, H., Guo, Y., and Jiang, X. (2020b). Characterizing code clones in the ethereum smart contract ecosystem. In *24th International Conference on Financial Cryptography and Data Security* (Springer International Publishing), vol. 12059 of *FC 2020, Lecture Notes in Computer Science (LNCS)*, 654–675. doi:10.1007/978-3-030-51280-4_35

Hegedus, P. (2018). Towards analyzing the complexity landscape of solidity based ethereum smart contracts. In *1st IEEE/ACM International Workshop on Emerging Trends in Software Engineering for Blockchain, WETSEB@ICSE 2018* (New York, NY, USA: Association for Computing Machinery), WETSEB 18, 35–39. doi:10.1145/3194113.3194119

Hildenbrandt, E., Saxena, M., Rodrigues, N., Zhu, X., Daian, P., Guth, D., et al. (2018). Kevm: A complete formal semantics of the ethereum virtual machine. In *IEEE 31st Computer Security Foundations Symposium* (IEEE), CSF 2018, 204–217. doi:10.1109/CSF.2018.00022

Hirai, Y. (2017). Defining the ethereum virtual machine for interactive theorem provers. In *Financial Cryptography and Data Security*, ed. B (Springer International Publishing), vol. 10323 of *FC 2017*, 520–535. doi:10.1007/978-3-319-70278-0_33

Honig, J. J., Everts, M. H., and Huisman, M. (2019). Practical mutation testing for smart contracts. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, eds. C. Pérez-Solà, G. Navarro-Arribas, A. Biryukov, and J. Garcia-Alfaro (Cham: Springer International Publishing), 289–303. doi:10.1007/978-3-030-31500-9_19

Hu, B., Zhang, Z., Liu, J., Liu, Y., Yin, J., Lu, R., et al. (2021a). A comprehensive survey on smart contract construction and execution: paradigms, tools, and systems. *Patterns* 2, 100179. doi:10.1016/j.patter.2020.100179

Hu, T., Liu, X., Chen, T., Zhang, X., Huang, X., Niu, W., et al. (2021b). Transaction-based classification and detection approach for ethereum smart contract. *Information Processing & Management* 58, 102462. doi:10.1016/j.ipm.2020.102462

Huang, J., Han, S., You, W., Shi, W., Liang, B., Wu, J., et al. (2021). Hunting vulnerable smart contracts via graph embedding based bytecode matching. *IEEE Transactions on Information Forensics and Security* , 1–1doi:10.1109/TIFS.2021.3050051

Huang, T. H.-D. (2018). Hunting the ethereum smart contract: Color-inspired inspection of potential attacks. *arXiv preprint arXiv:1807.01868*

Huang, Y., Bian, Y., Li, R., Zhao, J. L., and Shi, P. (2019). Smart contract security: A software lifecycle perspective. *IEEE Access* 7, 150184–150202. doi:10.1109/access.2019.2946988

Hwang, S. and Ryu, S. (2020). Gap between theory and practice: An empirical study of security patches in solidity. In *IEEE/ACM 42nd International Conference on Software Engineering (ICSE)* (Association for Computing Machinery), ICSE 20, 542–553. doi:10.1145/3377811.3380424

Imeri, A., Agoulmine, N., and Khadraoui, D. (2020). Smart contract modeling and verification techniques: A survey. In *8th International Workshop on ADVANCEs in ICT Infrastructures and Services (ADVANCE 2020)* (Candy E. Sansores, Universidad del Caribe, Mexico, Nazim Agoulmine, IBISC Lab, University of Evry - Paris-Saclay University), ADVANCE 2020, 1–8

Ivanova, Y. and Khritankov, A. (2020). Regularmutator: A mutation testing tool for solidity smart contracts. *Procedia Computer Science, 9th International Young Scientists Conference in Computational Science* 178, 75–83. doi:10.1016/j.procs.2020.11.009

Jiang, B., Liu, Y., and Chan, W. K. (2018). Contractfuzzer: Fuzzing smart contracts for vulnerability detection. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering* (Association for Computing Machinery), ASE 2018, 259–269. doi:10.1145/3238147.3238177

Jiao, J., Kan, S., Lin, S.-W., Sanan, D., Liu, Y., and Sun, J. (2020). Semantic understanding of smart contracts: Executable operational semantics of solidity. In *IEEE Symposium on Security and Privacy (SP)* (IEEE), SP 2020, 1695–1712. doi:10.1109/SP40000.2020.00066

Kaleem, M., Mavridou, A., and Laszka, A. (2020). Vyper: A security comparison with solidity based on common vulnerabilities. In *2nd Conference on Blockchain Research Applications for Innovative Networks and Services (BRAINS)* (IEEE), BRAINS 20, 107–111. doi:10.1109/BRAINS49436.2020.9223278

Kalra, S., Goel, S., Dhawan, M., and Sharma, S. (2018). Zeus: Analyzing safety of smart contracts. In *Network and Distributed Systems Security Symposium*. NDSS 2018, 1–15. doi:10.14722/ndss.2018.23082

Khan, Z. A. and Namin, A. S. (2020). A survey on vulnerabilities of ethereum smart contracts. *arXiv preprint arXiv:2012.14481*

Khor, J., Masama, M. A., Sidorov, M., Leong, W., and Lim, J. (2020). An improved gas efficient library for securing iot smart contracts against arithmetic vulnerabilities. In *Proceedings of the 2020 9th International Conference on Software and Computer Applications* (Association for Computing Machinery), ICSCA 2020, 326–330. doi:10.1145/3384544.3384577

Kim, K. B. and Lee, J. (2020). Automated generation of test cases for smart contract security analyzers. *IEEE Access* 8, 209377–209392. doi:10.1109/access.2020.3039990

Kim, S. and Ryu, S. (2020). Analysis of blockchain smart contracts: Techniques and insights. In *IEEE Secure Development (SecDev)* (IEEE), SecDev 2020, 65–73. doi:10.1109/secdev45635.2020.00026

Kolluri, A., Nikolic, I., Sergey, I., Hobor, A., and Saxena, P. (2019). Exploiting the laws of order in smart contracts. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis* (New York, NY, USA: Association for Computing Machinery), ISSTA 2019, 363–373. doi:10.1145/3293882.3330560

Krupa, T., Ries, M., Kotuliak, I., Kostal, K., and Bencel, R. (2020). Security issues of smart contracts in ethereum platforms. *Preprint* doi:10.13140/RG.2.2.17996.85128

Krupp, J. and Rossow, C. (2018). teether: Gnawing at ethereum to automatically exploit smart contracts. In *Proceedings of the 27th USENIX Conference on Security Symposium* (USA: USENIX Association), USENIX Security 2018, 1317–1333

Lagouvardos, S., Grech, N., Tsatiris, I., and Smaragdakis, Y. (2020). Precise static modeling of ethereum 'memory'. *Proceedings of the ACM on Programming Languages* 4, 26. doi:10.1145/3428258

Lee, W. Y. and Choi, Y.-S. (2020). Vulnerability and cost analysis of heterogeneous smart contract programs in blockchain systems. *Current Trends in Computer Sciences and Applications* 2, 142–145. doi:10.32474/ctcsa.2020.02.000126

Leid, A., van der Merwe, B., and Visser, W. (2020). Testing ethereum smart contracts: A comparison of symbolic analysis and fuzz testing tools. In *Conference of South African Institute of Computer Scientists and Information Technologists* (Association for Computing Machinery), SAICSIT 20, 35–43. doi:10.1145/3410886.3410907

Li, A., Choi, J. A., and Long, F. (2020). Securing smart contract with runtime validation. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation* (New York, NY, USA: Association for Computing Machinery), PLDI 2020, 438–453. doi:10.1145/3385412.3385982

Li, A. and Long, F. (2019). Detecting standard violation errors in smart contracts. *arXiv preprint arXiv:1812.07702*

Li, X., Chen, T., Luo, X., Zhang, T., Yu, L., and Xu, Z. (2020). Stan: Towards describing bytecodes of smart contract. In *IEEE 20th International Conference on Software Quality, Reliability and Security* (IEEE), QRS 20, 273–284. doi:10.1109/QRS51102.2020.00045

Li, X., Jiang, P., Chen, T., Luo, X., and Wen, Q. (2020). A survey on the security of blockchain systems. *Future Generation Computer Systems* 107, 841–853. doi:10.1016/j.future.2017.08.020

Li, Y. (2019). Finding concurrency exploits on smart contracts. In *Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings* (IEEE), ICSE 19, 144–146. doi:10.1109/ICSE-Companion.2019.00061

Li, Z., Wu, H., Xu, J., Wang, X., Zhang, L., and Chen, Z. (2019). Musc: A tool for mutation testing of ethereum smart contract. In *34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE (IEEE), ASE 19, 1198–1201. doi:10.1109/ase.2019.00136

Liao, J.-W., Tsai, T.-T., He, C.-K., and Tien, C.-W. (2019). Soliaudit: Smart contract vulnerability assessment based on machine learning and fuzz testing. In *Sixth International Conference on Internet of Things: Systems, Management and Security* (IEEE), IoTSMS 2019, 458–465. doi:10.1109/iotsms48152.2019.8939256

Liu, C., Liu, H., Cao, Z., Chen, Z., Chen, B., and Roscoe, B. (2018a). Reguard: Finding reentrancy bugs in smart contracts. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings* (Association for Computing Machinery), ICSE 18, 65–68. doi:10.1145/3183440.3183495

Liu, H., Liu, C., Zhao, W., Jiang, Y., and Sun, J. (2018b). S-gram: Towards semantic-aware security auditing for ethereum smart contracts. In *33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE (Association for Computing Machinery), ASE 18, 814–819. doi:10.1145/3238147.3240728

Liu, H., Yang, Z., Jiang, Y., Zhao, W., and Sun, J. (2019). Enabling clone detection for ethereum via smart contract birthmarks. In *IEEE/ACM 27th International Conference on Program Comprehension (ICPC)* (IEEE), 105–115. doi:10.1109/icpc.2019.00024

Liu, Y., Li, Y., Lin, S.-W., and Yan, Q. (2020a). Modcon: A model-based testing platform for smart contracts. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Association for Computing Machinery), ESEC/FSE 2020, 1601–1605. doi:10.1145/3368089.3417939

Liu, Y., Li, Y., Lin, S.-W., and Zhao, R. (2020b). Towards automated verification of smart contract fairness. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Association for Computing Machinery), ESEC/FSE 2020, 666–677. doi:10.1145/3368089.3409740

Livshits, B. (2020). Technical perspective: Analyzing smart contracts with madmax. *Communications of the ACM* 63, 86. doi:10.1145/3416259

López Vivar, A., Castedo, A. T., Sandoval Orozco, A. L., and Garcia Villalba, L. J. (2020). An analysis of smart contracts security threats alongside existing solutions. *Entropy* 22, 203. doi:10.3390/e22020203

Lu, N., Wang, B., Zhang, Y., Shi, W., and Esposito, C. (2019). Neucheck: A more practical ethereum smart contract security analysis tool. *Software: Practice and Experience* doi:10.1002/spe.2745

Luu, L., Chu, D.-H., Olickel, H., Saxena, P., and Hobor, A. (2016). Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Association for Computing Machinery), CCS 16, 254–269. doi:10.1145/2976749.2978309

Ma, F., Fu, Y., Ren, M., Wang, M., Jiang, Y., Zhang, K., et al. (2019). Evm*: From offline detection to online reinforcement for ethereum virtual machine. In *IEEE 26th International Conference on Software Analysis, Evolution and Reengineering* (IEEE), SANER 19, European Conference on Software Maintenance and Reengineering (CSMR), 554–558. doi:10.1109/saner.2019.8668038

Magazzeni, D., McBurney, P., and Nash, W. (2017). Validation and verification of smart contracts: A research agenda. *Computer* 50, 50–57. doi:10.1109/mc.2017.3571045

Marescotti, M., Otoni, R., Alt, L., Eugster, P., Hyvärinen, A. E. J., and Sharygina, N. (2020). Accurate smart contract verification through direct modelling. In *Leveraging Applications of Formal Methods, Verification and Validation: Applications*, eds. T. Margaria and B. Steffen (Springer International Publishing), vol. 12478 of *ISoLA 2020, Lecture Notes in Computer Science (LNCS)*, 178–194. doi:10.1007/978-3-030-61467-6_12

Mavridou, A. and Laszka, A. (2018a). Designing secure ethereum smart contracts: A finite state machine based approach. In *Financial Cryptography and Data Security*, ed. K. Meiklejohn, Sarahand Sako (Springer Berlin Heidelberg), vol. 10957 of *Lecture Notes in Computer Science LNCS*, 523–540. doi:10.1007/978-3-662-58387-6_28

Mavridou, A. and Laszka, A. (2018b). Tool demonstration: Fsolidm for designing secure ethereum smart contracts. In *International Conference on Principles of Security and Trust* (Springer International Publishing), vol. 10804 of *POST 2018, Lecture Notes in Computer Science (LNCS)*, 270–277. doi:10.1007/978-3-319-89722-6_11

Mavridou, A., Laszka, A., Stachtiari, E., and Dubey, A. (2019). Verisolid: Correct-by-design smart contracts for ethereum. In *Financial Cryptography and Data Security*, eds. I. Goldberg and T. Moore (Cham: Springer International Publishing), vol. 11598 of *FC 2019*, 446–465. doi:10.1007/978-3-030-32101-7_27

Mei, X., Ashraf, I., Jiang, B., and Chan, W. K. (2019). A fuzz testing service for assuring smart contracts. In *IEEE 19th International Conference on Software Quality, Reliability and Security Companion* (IEEE), QRS 19, 544–545. doi:10.1109/QRS-C.2019.00116

Mense, A. and Flatscher, M. (2018). Security vulnerabilities in ethereum smart contracts. In *Proceedings of the 20th International Conference on Information Integration and Web-Based Applications & Services* (Association for Computing Machinery), iiWAS 2018, 375–380. doi:10.1145/3282373.3282419

Miller, A., Cai, Z., and Jha, S. (2018). Smart contracts and opportunities for formal methods. In *Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice*, eds. T. Margaria and B. Steffen (Cham: Springer International Publishing), vol. 11247 of *ISoLA 2018, Lecture Notes in Computer Science (LNCS)*, 280–299. doi:10.1007/978-3-030-03427-6_22

Min, T. and Cai, W. (2019). A security case study for blockchain games. In *IEEE Games, Entertainment, Media Conference (GEM)* (IEEE), 1–8. doi:10.1109/GEM.2019.8811555

Moona, A. and Mathew, R. (2021). Review of tools for analyzing security vulnerabilities in ethereum based smart contracts. In *International Conference on IoT Based Control Networks & Intelligent Systems* (SSRN), ICICNIS 2020, 1–10. doi:10.2139/ssrn.3769774

Mossberg, M., Manzano, F., Hennenfent, E., Groce, A., Grieco, G., Feist, J., et al. (2019). Manticore: A user-friendly symbolic execution framework for binaries and smart contracts. In *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering* (IEEE), ASE 19, 1186–1189. doi:10.1109/ASE.2019.00133

Mueller, B. (2018). Smashing ethereum smart contracts for fun and real profit. In *The 9th annual HITB Security Conference* (ConsenSys Diligence), HITB SECCONF 2018, 54 pages

Murray, Y. and Anisi, D. A. (2019). Survey of formal verification methods for smart contracts on blockchain. In *10th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*. IEEE (IEEE), IFIP NTMS 19, 1–6. doi:10.1109/ntms.2019.8763832

[Dataset] NCC Group (2018). Decentralized application security project (DASP) top 10. `https://dasp.co`. Accessed: 2021-08-08

Nehaï, Z., Piriou, P.-Y., and Daumas, F. (2018). Model-checking of smart contracts. In *IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)* (IEEE), 980–987. doi:10.1109/Cybermatics_2018.2018.00185

Nehaï, Z. and Bobot, F. (2019). Deductive proof of industrial smart contracts using why3. In *Formal Methods. FM 2019 International Workshops* (Springer International Publishing), vol. 12232 of *FM 2019, Lecture Notes in Computer Science (LNCS)*, 299–311. doi:10.1007/978-3-030-54994-7_22

Nelaturu, K., Mavridoul, A., Veneris, A., and Laszka, A. (2020). Verified development and deployment of multiple interacting smart contracts with verisolid. In *IEEE International Conference on Blockchain and Cryptocurrency (ICBC)* (IEEE), 1–9. doi:10.1109/ICBC48266.2020.9169428

Nguyen, Q.-B., Nguyen, A.-Q., Nguyen, V.-H., Nguyen-Le, T., and Nguyen-An, K. (2019). Detect abnormal behaviours in ethereum smart contracts using attack vectors. In *6th International Conference on Future Data and Security Engineering* (Springer International Publishing), vol. 11814 of *FDSE 2019, Lecture Notes in Computer Science (LNCS)*, 485–505. doi:10.1007/978-3-030-35653-8_32

Nguyen, T. D., Pham, L. H., Sun, J., Lin, Y., and Minh, Q. T. (2020). Sfuzz: An efficient adaptive fuzzer for solidity smart contracts. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering* (New York, NY, USA: Association for Computing Machinery), ICSE 20, 778–788. doi:10.1145/3377811.3380334

Nikolić, I., Kolluri, A., Sergey, I., Saxena, P., and Hobor, A. (2018). Finding the greedy, prodigal, and suicidal contracts at scale. In *Proceedings of the 34th Annual Computer Security Applications Conference* (New York, NY, USA: Association for Computing Machinery), ACSAC 18, 653–663. doi:10.1145/3274694.3274743

Norvill, R., Pontiveros, B. B. F., State, R., and Cullen, A. (2018). Visual emulation for ethereum's virtual machine. In *IEEE/IFIP Network Operations and Management Symposium* (IEEE), NOMS 2018, 1–4. doi:10.1109/NOMS.2018.8406332

Osterland, T. and Rose, T. (2020). Model checking smart contracts for ethereum. *Pervasive and Mobile Computing* 63, 101129. doi:10.1016/j.pmcj.2020.101129

Pankov, K. N. (2020). Testing, verification and validation of distributed ledger systems. In *Systems of Signals Generating and Processing in the Field of on Board Communications* (IEEE), 1–9. doi:10.1109/IEEECONF48371.2020.9078541

Parizi, R. M., Dehghantanha, A., Choo, K.-K. R., and Singh, A. (2018). Empirical vulnerability analysis of automated smart contracts security testing on blockchains. In *Proceedings of the 28th Annual International Conference on Computer Science and Software Engineering* (IBM Corp.), CASCON 18, 103–113

Park, D., Zhang, Y., and Rosu, G. (2020). End-to-end formal verification of ethereum 2.0 deposit smart contract. In *International Conference on Computer Aided Verification* (Springer International Publishing), vol. 12224 of *CAV 2020, Lecture Notes in Computer Science (LNCS)*, 151–164. doi:10.1007/978-3-030-53288-8_8

Park, D., Zhang, Y., Saxena, M., Daian, P., and Roşu, G. (2018). A formal verification tool for ethereum vm bytecode. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Association for Computing Machinery), ESEC/FSE 2018, 912–915. doi:10.1145/3236024.3264591

Peng, C., Akca, S., and Rajan, A. (2019). Sif: A framework for solidity contract instrumentation and analysis. In *26th Asia-Pacific Software Engineering Conference (APSEC)* (IEEE), APSEC 19, 466–473. doi:10.1109/APSEC48747.2019.00069

Perez, D. and Livshits, B. (2019). Smart contract vulnerabilities: Vulnerable does not imply exploited. *arXiv preprint arXiv:1902.06710*

Pérez, V., Klemen, M., López-García, P., Morales, J. F., and Hermenegildo, M. (2020). Cost analysis of smart contracts via parametric resource analysis. In *International Static Analysis Symposium*, eds. D. Pichardie and M. Sighireanu (Springer International Publishing), vol. 12389 of *SAS 2020, Lecture Notes in Computer Science(LNCS)*, 7–31. doi:10.1007/978-3-030-65474-0_2

Permenev, A., Dimitrov, D., Tsankov, P., Drachsler-Cohen, D., and Vechev, M. (2020). Verx: Safety verification of smart contracts. In *IEEE Symposium on Security and Privacy (SP)* (IEEE), SP 2020, 1661–1677. doi:10.1109/sp40000.2020.00024

Praitheeshan, P., Pan, L., and Doss, R. (2020a). Security evaluation of smart contract-based on-chain ethereum wallets. In *International Conference on Network and System Security*, eds. M. Kutyłowski, J. Zhang, and C. Chen (Springer International Publishing), vol. 12570 of *NSS 2020, Lecture Notes in Computer Science (LNCS)*, 22–41. doi:10.1007/978-3-030-65745-1_2

Praitheeshan, P., Pan, L., Yu, J., Liu, J., and Doss, R. (2020b). Security analysis methods on ethereum smart contract vulnerabilities: A survey. *arXiv preprint arXiv:1908.08605v3*

Prechtel, D., Groß, T., and Müller, T. (2019). Evaluating spread of 'gasless send' in ethereum smart contracts. In *10th IFIP International Conference on New Technologies, Mobility and Security (NTMS)* (IEEE), IFIP NTMS 19, 1–6. doi:10.1109/NTMS.2019.8763848

Qasse, I. A., Spillner, J., Abu Talib, M., and Nasir, Q. (2020). A study on DApps characteristics. In *2020 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)* (IEEE), DAPPS 2020, 88–93. doi:10.1109/DAPPS49028.2020.00010

Qian, P., Liu, Z., He, Q., Zimmermann, R., and Wang, X. (2020). Towards automated reentrancy detection for smart contracts based on sequential models. *IEEE Access* 8, 19685–19695. doi:10.1109/access.2020.2969429

Quan, L., Wu, L., and Wang, H. (2019). Evulhunter: Detecting fake transfer vulnerabilities for eosio's smart contracts at webassembly-level. *arXiv preprint arXiv:1906.10362*

Radu Adrian, O. (2018). The blockchain, today and tomorrow. In *20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. SYNASC 18, 458–462. doi:10.1109/SYNASC.2018.00077

Rodler, M., Li, W., Karame, G. O., and Davi, L. (2018). Sereum: Protecting existing smart contracts against re-entrancy attacks. *arXiv preprint arXiv:1812.05934*

Saad, M., Spaulding, J., Njilla, L., Kamhoua, C., Shetty, S., Nyang, D., et al. (2020). Exploring the attack surface of blockchain: A comprehensive survey. *IEEE Communications Surveys Tutorials* 22, 1977–2008. doi:10.1109/COMST.2020.2975999

Samreen, N. F. and Alalfi, M. H. (2020a). Reentrancy vulnerability identification in ethereum smart contracts. In *IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE)* (IEEE), 22–29. doi:10.1109/iwbose50093.2020.9050260

Samreen, N. F. and Alalfi, M. H. (2020b). A survey of security vulnerabilities in ethereum smart contracts. In *Proceedings of the 30th Annual International Conference on Computer Science and Software Engineering* (IBM Corp.), CASCON 20, 73–82

Sayeed, S., Marco-Gisbert, H., and Caira, T. (2020). Smart contract: Attacks and protections. *IEEE Access* 8, 24416–24427. doi:10.1109/ACCESS.2020.2970495

Schneidewind, C., Grishchenko, I., Scherer, M., and Maffei, M. (2020a). ethor: Practical and provably sound static analysis of ethereum smart contracts. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (Association for Computing Machinery), CCS 20, 621–640. doi:10.1145/3372297.3417250

Schneidewind, C., Scherer, M., and Maffei, M. (2020b). The good, the bad and the ugly: Pitfalls and best practices in automated sound static analysis of ethereum smart contracts. In *International Symposium on Leveraging Applications of Formal Methods, Verification and Validation: Applications*, eds. T. Margaria and B. Steffen (Springer International Publishing), vol. 12478 of *ISoLA 2020, Lecture Notes in Computer Science (LNCS)*, 212–231. doi:10.1007/978-3-030-61467-6_14

Shishkin, E. (2019). Debugging smart contract's business logic using symbolic model checking. *Programming and Computer Software* 45, 590–599. doi:10.1134/S0361768819080164

Shmatko, A. and Olkhovskyi, D. (2020). Smart contract security problem review. *Scientific Collection InterConf* 3

Shrivas, M. K., Yeboah, T., and Brunda, S. S. (2020). Hybrid security framework for blockchain platforms. In *First International Conference on Power, Control and Computing Technologies (ICPC2T)* (IEEE), ICPC2T 20, 339–347. doi:10.1109/ICPC2T48082.2020.9071477

Signer, C. (2018). *Gas cost analysis for ethereum smart contracts*. Master's thesis, ETH Zurich, Department of Computer Science. doi:10.3929/ethz-b-000312914

Singh, A., Parizi, R. M., Zhang, Q., Choo, K.-K. R., and Dehghantanha, A. (2020). Blockchain smart contracts formalization: Approaches and challenges to address vulnerabilities. *Computers & Security* 88, 101654. doi:10.1016/j.cose.2019.101654

So, S., Lee, M., Park, J., Lee, H., and Oh, H. (2020). Verismart: A highly precise safety verifier for ethereum smart contracts. In *IEEE Symposium on Security and Privacy (SP)* (IEEE), SP 2020, 1678–1694. doi:10.1109/SP40000.2020.00032

Staderini, M. and Palli, C. (2020). Analysis on ethereum vulnerabilities and further steps. In *Proceedings of the 27th PhD Mini-Symposium of the Department of Measurement and Information Systems, Budapest University of Technology and Economics* (RCL Resilient Computing Lab Research Group), 21–24

Staderini, M., Palli, C., and Bondavalli, A. (2020). Classification of ethereum vulnerabilities and their propagations. In *Second International Conference on Blockchain Computing and Applications* (IEEE), BCCA 2020, 44–51. doi:10.1109/bcca50787.2020.9274458

[Dataset] Stortz, R. (2018). Rattle–an ethereum evm binary analysis framework. `https://www.trailofbits.com/presentations/rattle/`. Accessed: 2021-08-08

Suvorov, D. and Ulyantsev, V. (2019). Smart contract design meets state machine synthesis: Case studies. *arXiv preprint arXiv:1906.02906*

[Dataset] SWC Registry (2018). Smart contract weakness classification and test cases. `https://swcregistry.io/`. Accessed: 2021-08-08

Tan, S., S Bhowmick, S., Chua, H. E., and Xiao, X. (2020). Latte: Visual construction of smart contracts. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA: Association for Computing Machinery), SIGMOD 20, 2713–2716. doi:10.1145/3318464.3384687

Tann, W. J.-W., Han, X. J., Gupta, S. S., and Ong, Y.-S. (2018). Towards safer smart contracts: A sequence learning approach to detecting security threats. *arXiv preprint arXiv:1811.06632*

Tantikul, P. and Ngamsuriyaroj, S. (2020). Exploring vulnerabilities in solidity smart contract. In *Proceedings of the 6th International Conference on Information Systems Security and Privacy* (SciTePress), vol. 1 of *ICISSP 2020*, 317–324. doi:10.5220/0008909803170324

Tikhomirov, S., Voskresenskaya, E., Ivanitskiy, I., Takhaviev, R., Marchenko, E., and Alexandrov, Y. (2018). Smartcheck: Static analysis of ethereum smart contracts. In *1st IEEE/ACM International Workshop on Emerging Trends in Software Engineering for Blockchain, WETSEB@ICSE 2018* (Association for Computing Machinery), WETSEB 18, 9–16. doi:10.1145/3194113.3194115

Tolmach, P., Li, Y., Lin, S.-W., Liu, Y., and Li, Z. (2020). A survey of smart contract formal specification and verification. *arXiv preprint arXiv:2008.02712*

Torres, C. F., Schütte, J., and State, R. (2018). Osiris: Hunting for integer bugs in ethereum smart contracts. In *Proceedings of the 34th Annual Computer Security Applications Conference* (Association for Computing Machinery), ACSAC 18, 664–676. doi:10.1145/3274694.3274737

Torres, C. F., Steichen, M., and State, R. (2019). The art of the scam: Demystifying honeypots in ethereum smart contracts. In *Proceedings of the 28th USENIX Conference on Security Symposium* (USA: USENIX Association), USENIX Security 2019, 1591–1607

Tsankov, P. (2018). Security analysis of smart contracts in datalog. In *International Symposium on Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice*, eds. T. Margaria and B. Steffen (Springer International Publishing), vol. 11247 of *ISoLA 2018, Lecture Notes in Computer Science (LNCS)*, 316–322. doi:10.1007/978-3-030-03427-6_24

Tsankov, P., Dan, A., Drachsler-Cohen, D., Gervais, A., Buenzli, F., and Vechev, M. (2018). Securify: Practical security analysis of smart contracts. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (Association for Computing Machinery), CCS 18, 67–82. doi:10.1145/3243734.3243780

Vacca, A., Di Sorbo, A., Visaggio, C. A., and Canfora, G. (2020). A systematic literature review of blockchain and smart contract development: Techniques, tools, and open challenges. *Journal of Systems and Software* 174, 110891. doi:10.1016/j.jss.2020.110891

Vandenbogaerde, B. (2019). A graph-based framework for analysing the design of smart contracts. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (New York, NY, USA: Association for Computing Machinery), ESEC/FSE 2019, 1220–1222. doi:10.1145/3338906.3342495

Vinayak, M., Pal Singh Panesar, H. A., Santos, S. d., Thulasiram, R. K., Thulasiraman, P., and Appadoo, S. S. (2018). Analyzing financial smart contracts for blockchain. In *IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)* (IEEE), iThings/GreenCom/CPSCom/SmartData 2018, 1701–1706. doi:10.1109/Cybermatics_2018.2018.00284

Wang, H., Li, Y., Lin, S.-W., Ma, L., and Liu, Y. (2019a). Vultron: Catching vulnerable smart contracts once and for all. In *Proceedings of the 41st International Conference on Software Engineering: New Ideas and Emerging Results* (IEEE), ICSE (NIER) 19, 1–4. doi:10.1109/ICSE-NIER.2019.00009

Wang, H., Liu, Y., Li, Y., Lin, S.-W., Artho, C., Ma, L., et al. (2020a). Oracle-supported dynamic exploit generation for smart contracts. *IEEE Transactions on Dependable and Secure Computing* , 1–1doi:10.1109/TDSC.2020.3037332

Wang, S., Zhang, C., and Su, Z. (2019b). Detecting nondeterministic payment bugs in ethereum smart contracts. *Proceedings of the ACM on Programming Languages (PACMPL)* 3, 1–29. doi:10.1145/3360615

Wang, W., Song, J., Xu, G., Li, Y., Wang, H., and Su, C. (2020b). Contractward: Automated vulnerability detection models for ethereum smart contracts. *IEEE Transactions on Network Science and Engineering* , 1–1doi:10.1109/tnse.2020.2968505

Wang, X., He, J., Xie, Z., Zhao, G., and Cheung, S.-C. (2019c). Contractguard: Defend ethereum smart contracts with embedded intrusion detection. *IEEE Transactions on Services Computing* 13, 314–328. doi:10.1109/tsc.2019.2949561

Wang, Y., Lahiri, S. K., Chen, S., Pan, R., Dillig, I., Born, C., et al. (2018). Formal specification and verification of smart contracts for azure blockchain. *arXiv preprint arXiv:1812.08829*

Wang, Y., Lahiri, S. K., Chen, S., Pan, R., Dillig, I., Born, C., et al. (2020c). Formal verification of workflow policies for smart contracts in azure blockchain. In *Verified Software. Theories, Tools, and Experiments*, eds. S. Chakraborty and J. A. Navas (Springer International Publishing), vol. 12031 of *VSTTE 2019, Lecture Notes in Computer Science (LNCS)*, 87–106. doi:10.1007/978-3-030-41600-3_7

Wang, Z., Dai, W., Choo, K.-K. R., Jin, H., and Zou, D. (2020d). Fsfc: An input filter-based secure framework for smart contract. *Journal of Network and Computer Applications* 154, 102530. doi:10.1016/j.jnca.2020.102530

Weiss, K. and Schütte, J. (2019). Annotary: A concolic execution system for developing secure smart contracts. In *24th European Symposium On Research In Computer Security*, eds. K. Sako, S. Schneider, and P. Y. A. Ryan (Cham: Springer International Publishing), ESORICS 2019, 747–766. doi:10.1007/978-3-030-29959-0_36

Wohrer, M. and Zdun, U. (2018). Smart contracts: security patterns in the ethereum ecosystem and solidity. In *International Workshop on Blockchain Oriented Software Engineering (IWBOSE)* (IEEE), 2–8. doi:10.1109/iwbose.2018.8327565

Wüstholz, V. and Christakis, M. (2020a). Harvey: A greybox fuzzer for smart contracts. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Association for Computing Machinery), ESEC/FSE 2020, 1398–1409. doi:10.1145/3368089.3417064

Wüstholz, V. and Christakis, M. (2020b). Targeted greybox fuzzing with static lookahead analysis. In *IEEE/ACM 42nd International Conference on Software Engineering (ICSE)* (IEEE), ICSE 20, 789–800. doi:10.1145/3377811.3380388

Wu, L., Wu, S., Zhou, Y., Li, R., Wang, Z., Luo, X., et al. (2020). Time-travel investigation: Towards building a scalable attack detection framework on ethereum. *arXiv preprint arXiv:2005.08278*

Xu, J., Dang, F., Ding, X., and Zhou, M. (2020). A survey on vulnerability detection tools of smart contract bytecode. In *IEEE 3rd International Conference on Information Systems and Computer Aided Education* (IEEE), ICISCAE 20, 94–98. doi:10.1109/iciscae51034.2020.9236931

Xue, Y., Ma, M., Lin, Y., Sui, Y., Ye, J., and Peng, T. (2020). Cross-contract static analysis for detecting practical reentrancy vulnerabilities in smart contracts. In *35th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (ACM), ASE 20, 1029–1040. doi:10.1145/3324884.3416553

Yamashita, K., Nomura, Y., Zhou, E., Pi, B., and Jun, S. (2019). Potential risks of hyperledger fabric smart contracts. In *IEEE 2nd International Workshop on Blockchain Oriented Software Engineering*

*(IWBOSE)* (IEEE), 1–10. doi:10.1109/IWBOSE.2019.8666486

Yang, Z., Keung, J., Zhang, M., Xiao, Y., Huang, Y., and Hui, T. (2020a). Smart contracts vulnerability auditing with multi-semantics. In *IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)* (IEEE), 892–901. doi:10.1109/compsac48688.2020.0-153

Yang, Z. and Lei, H. (2018a). Formal process virtual machine for smart contracts verification. *International Journal of Performability Engineering* 14, 1726–1734. doi:10.23940/ijpe.18.08.p9.17261734

Yang, Z. and Lei, H. (2018b). Optimization of executable formal interpreters developed in higher-order logic theorem proving systems. *IEEE Access* 6, 70331–70348. doi:10.1109/ACCESS.2018.2880692

Yang, Z. and Lei, H. (2019a). Fether: An extensible definitional interpreter for smart-contract verifications in coq. *IEEE Access* 7, 37770–37791. doi:10.1109/ACCESS.2019.2905428

Yang, Z. and Lei, H. (2019b). A general formal memory framework for smart contracts verification based on higher-order logic theorem proving. *International Journal of Performability Engineering* 15, 2998–3007. doi:10.23940/ijpe.19.11.p19.29983007

Yang, Z., Lei, H., and Qian, W. (2020b). A hybrid formal verification system in coq for ensuring the reliability and security of ethereum-based service smart contracts. *IEEE Access* 8, 21411–21436. doi:10.1109/access.2020.2969437

Yang, Z., Liu, H., Li, Y., Zheng, H., Wang, L., and Chen, B. (2020c). Seraph: Enabling cross-platform security analysis for evm and wasm smart contracts. In *ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings* (New York, NY, USA: Association for Computing Machinery), ICSE 20, 21–24. doi:10.1145/3377812.3382157

Ye, J., Ma, M., Lin, Y., Sui, Y., and Xue, Y. (2020). Clairvoyance: Cross-contract static analysis for detecting practical reentrancy vulnerabilities in smart contracts. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings*. IEEE (Association for Computing Machinery), ICSE 20, 274–275. doi:10.1145/3377812.3390908

Ye, J., Ma, M., Peng, T., Peng, Y., and Xue, Y. (2019a). Towards automated generation of bug benchmark for smart contracts. In *IEEE International Conference on Software Testing, Verification and Validation Workshops* (IEEE), ICST Workshops 2019, 184–187. doi:10.1109/icstw.2019.00049

Ye, J., Ma, M., Peng, T., and Xue, Y. (2019b). A software analysis based vulnerability detection system for smart contracts. In *Integrating Research and Practice in Software Engineering*, eds. S. Jarzabek, A. Poniszewska-Marańda, and L. Madeyski (Springer International Publishing), vol. 851 of *Studies in Computational Intelligence, SCI*. 69–81. doi:10.1007/978-3-030-26574-8_6

Yu, X. L., Al-Bataineh, O., Lo, D., and Roychoudhury, A. (2020). Smart contract repair. *ACM Transactions on Software Engineering and Methodology* 29. doi:10.1145/3402450

Zakrzewski, J. (2018). Towards verification of ethereum smart contracts: A formalization of core of solidity. In *Working Conference on Verified Software: Theories, Tools, and Experiments*, eds. R. Piskac and P. Rümmer (Springer International Publishing), VSTTE 2018, Lecture Notes in Computer Science (LNCS), 229–247. doi:10.1007/978-3-030-03592-1_13

Zhang, F., Cecchetti, E., Croman, K., Juels, A., and Shi, E. (2016). Town crier: An authenticated data feed for smart contracts. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA: Association for Computing Machinery), CCS 16, 270–282. doi:10.1145/2976749.2978326

Zhang, P., Xiao, F., and Luo, X. (2019a). Solidity check: Quickly detecting smart contract problems through regular expressions. *arXiv preprint arXiv:1911.09425*

Zhang, P., Xiao, F., and Luo, X. (2020a). A framework and dataset for bugs in ethereum smart contracts. In *IEEE International Conference on Software Maintenance and Evolution (ICSME)* (IEEE), ICSME

2020, 139–150. doi:10.1109/icsme46990.2020.00023

Zhang, P., Yu, J., and Ji, S. (2020b). Adf-ga: Data flow criterion based test case generation for ethereum smart contracts. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops* (New York, NY, USA: Association for Computing Machinery), ICSE 20, 754–761. doi:10.1145/3387940.3391499

Zhang, Q., Wang, Y., Li, J., and Ma, S. (2020c). Ethploit: From fuzzing to efficient exploit generation against smart contracts. In *IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)* (IEEE), SANER 20, European Conference on Software Maintenance and Reengineering (CSMR), 116–126. doi:10.1109/SANER48275.2020.9054822

Zhang, W., Banescu, S., Pasos, L., Stewart, S., and Ganesh, V. (2019b). Mpro: Combining static and symbolic analysis for scalable testing of smart contract. In *IEEE 30th International Symposium on Software Reliability Engineering* (IEEE), ISSRE 2019, 456–462. doi:10.1109/issre.2019.00052

Zhang, Y., Ma, S., Li, J., Li, K., Nepal, S., and Gu, D. (2020d). Smartshield: Automatic smart contract protection made easy. In *IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)* (IEEE), SANER 20, European Conference on Software Maintenance and Reengineering (CSMR), 23–34. doi:10.1109/SANER48275.2020.9054825

Zhou, E., Hua, S., Pi, B., Sun, J., Nomura, Y., Yamashita, K., et al. (2018a). Security assurance for smart contract. In *9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)* (IEEE), NTMS 2018, 1–5. doi:10.1109/ntms.2018.8328743

Zhou, S., Yang, Z., Xiang, J., Cao, Y., Yang, Z., and Zhang, Y. (2020). An ever-evolving game: Evaluation of real-world attacks and defenses in ethereum ecosystem. In *29th USENIX Security Symposium (USENIX Security 20)* (USENIX Association), USENIX Security 2020, 2793–2810

Zhou, Y., Kumar, D., Bakshi, S., Mason, J., Miller, A., and Bailey, M. (2018b). Erays: Reverse engineering ethereumś opaque smart contracts. In *Proceedings of the 27th USENIX Conference on Security Symposium* (USA: USENIX Association), USENIX Security 2018, 1371–1385

Zhuang, Y., Liu, Z., Qian, P., Liu, Q., Wang, X., and He, Q. (2020). Smart contract vulnerability detection using graph neural network. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, ed. C. Bessiere (International Joint Conferences on Artificial Intelligence Organization), IJCAI 20, 3283–3290. doi:10.24963/ijcai.2020/454

Zou, W., Lo, D., Kochhar, P. S., Le, X.-B. D., Xia, X., Feng, Y., et al. (2019). Smart contract development: Challenges and opportunities. *IEEE Transactions on Software Engineering* , 1–1doi:10.1109/TSE.2019.2942301

Zupan, N., Kasinathan, P., Cuellar, J., and Sauer, M. (2020). Secure smart contract generation based on petri nets. In *Blockchain Technology for Industry 4.0: Secure, Decentralized, Distributed and Trusted Industry Environment* (Springer Singapore), Blockchain Technologies (BT). 73–98. doi:10.1007/978-981-15-1137-0_4