Smart Contract Vulnerability Detection Methods: A Systematic Literature Review

André Storhaug

Fall 2021

Abstract

The `ntnuthesis` document class is a customised version of the standard LaTeX `report` document class. It can be used for theses at all levels – bachelor, master and PhD – and is available in English (British and American) and Norwegian (Bokmål and Nynorsk). This document is ment to serve (i) as a description of the document class, (ii) as an example of how to use it, and (iii) as a thesis template.

Sammendrag

Dokumentklassen `ntnuthesis` er en tilpasset versjon av LaTeX' standard `report`-klasse. Den er tilrettelagt for avhandlinger på alle nivåer – bachelor, master og PhD – og er tilgjengelig på både norsk (bokmål og nynorsk) og engelsk (britisk og amerikansk). Dette dokumentet er ment å tjene (i) som en beskrivelse av dokumentklassen, (ii) som et eksempel på bruken av den, og (iii) som en mal for avhandlingen.

Contents

Figures

Tables

Code Listings

Acronyms

ABI Application Binary Interface. 14, 18, 19

AST Abstract Syntax Tree. 17, 19, 20, 22

Att-BLSTM Attention-Based Bidirectional Long Short-Term Memory. 20, 21

CFG Control Flow Graph. 14, 15, 18–20

CNN Convolutional Neural Network. 20

DNN Deep Neural Network. 20, 21

GAN Generative Adversarial Network. 20, 21

IR Intermediate Representation. viii, 16–19, 23, *Glossary:* Intermediate Representation

KNN K-Nearest Neighbor. 22

LSTM Long Short-Term Memory. 20, 21

ML Machine Learning. 11, 23, 24

NFT Non Fungible Tokens. viii, 3, *Glossary:* Non Fungible Tokens

RF Random Forest. 22

SC Smart Contract. iv, vii, x, 2–5, 7, 8, 11, 14–19, 21–25

SGD Stochastic Gradient Descent. 22

SLR Systematic Literature Review. v, 7–9, 11

SMT Satisfiability Modulo Theories. viii, 15, *Glossary:* Satisfiability Modulo Theories

SVM Support Vector Machine. 22

TFIDF Term Frequency–Inverse Document Frequency. 21

WoS Web of Science. ix, 8, *Glossary:* Web of Science

Glossary

bytecode Bytecode is computer object code that is processed by a program (VM)s. 11

dapp A decentralized application is a computer application that runs on a decentralized computing system. 10

F* General-purpose functional programming language with effects aimed at program verification. 23

Intermediate Representation A representation for use as an intermediate step. 16–19, 23

LLVM A compiler toolchain providing a complete infrastructure for creating compiler frontends and backends. 17, 23

LLVM-IR LLVM-IR is the intermediate representation of the LLVM compiler toolchain. 17, 23

Non Fungible Tokens A type of token that is unique. 3

oracle Entities that connect blockchains to external systems, enabling smart contracts to execute based upon inputs and outputs from the real world. 18

Satisfiability Modulo Theories The problem of determining whether a mathematical formula is satisfiable. 15

Solidity Solidity is a Smart Contract language developed for Ethereum. 17, 23

Web of Science Scientific citation database. 8

Z3 A theorem prover (SMT solver) from Microsoft Research. 14

Todo list

Chapter 1

Introduction

Over the years, several thesis templates for LaTeX have been developed by different groups at NTNU. Typically, there have been local templates for given study programmes, or different templates for the different study levels – bachelor, master, and phd.[1]

Based on this experience, the CoPCSE[2] is hereby offering a template that should in principle be applicable for theses at all study levels. It is closely based on the standard LaTeX `report` document class as well as previous thesis templates. Since the central regulations for thesis design have been relaxed – at least for some of the historical university colleges now part of NTNU – the template has been simlified and put closer to the default LaTeX look and feel.

The purpose of the present document is threefold. It should serve (i) as a description of the document class, (ii) as an example of how to use it, and (iii) as a thesis template.

Cross-chain vulnerability detection is a method for detecting vulnerabilities in smart contract code across multiple blockchains. Little efforts have been made to develop a cross-chain vulnerability detection framework. We discuss the challenges and opportunities of cross-chain vulnerability detection.

call it paper or document?

The purpose of this document is to investigate the current state-of-the-art in cross-chain vulnerability detection. We will discuss the challenges and opportunities of cross-chain vulnerability detection. this review will have a broader scope (compared to other studies) than just Ethereum, including other blockchain platforms. Also, a focus will be on which vulnerabilities the tools are able to detect.

This rest of this paper is organized as follows. Chapter 2 describes the background of the project. The studies related to this litterature review are commented in Chapter 3. Chapter 4 describes the methods used to detect vulnerabilities. Chapter 5 describes the results of the project. Chapter 7 presents final remarks and concludes the paper.

[1] see, e.g., `https://github.com/COPCSE-NTNU/bachelor-thesis-NTNU` and `https://github.com/COPCSE-NTNU/master-theses-NTNU`

[2] `https://www.ntnu.no/wiki/display/copcse/Community+of+Practice+in+Computer+Science+Education+Home`

Chapter 2

Background

Research projects should always be based on previous research on the same and/or related topics. This should be described as a background to the thesis with adequate bibliographical references. If the material needed is too voluminous to fit nicely in the review part of the introduction, it can be presented in a separate background chapter.

2.1 Blockchain

A blockchain is a distributed ledger that is public and immutable. It is a record of transactions that can be verified by a number of parties. The blockchain is a public ledger that is distributed across the internet. Blockchain technology was popularized by Bitcoin in 2008. It enabled users to conduct transactions without the need for a central authority. From Bitcoin sprang several other cryptocurrencies and blockchain platforms such as Ethereum, Litecoin, and Ripple.

2.1.1 Ethereum

Ethereum is a decentralized platform that allows users to create, store, and transfer digital assets. It is a peer-to-peer network that operates without a central server. Ethereum introduces the concept of smart contracts, which are computer programs that run automatically on the blockchain. Smart contracts are used to automate the creation of new digital assets, such as tokens, and to create new business models.

Solidity is a programming language that is used to write smart contracts. Solidity is a subset of the Ethereum programming language.

2.2 Smart Contracts

The term "Smart Contract" was introduced with the Ethereum platform in 2014. A Smart Contract (SC) is a program that is executed on a blockchain, enabling non-

trusting parties to create an *agreement*. SCs have enabled several interesting new concepts, such as Non Fungible Tokens (NFT) and entirely new business models.

2.3 Smart Contract Security Vulnerabilities

There are many vulnerabilities in smart contracts that can be exploited by malicious actors. Throughout the last years, an increase in use of the Ethereum network has led to the development of smart contracts that are vulnerable to attacks. Due to the nature of blockchain technology, the attack surface of smart contracts is different from that of traditional computing systems.

Following is a list of the most common vulnerabilities in smart contracts:

2.3.1 Integer Overflow and Underflow

2.3.2 Transaction-Ordering Dependence

In blockchain systems there is no guarantee on the execution order of transactions.

2.3.3 Stack Size Limit

2.3.4 Timestamp Dependency

Considering changing to "Block state dependence"

Code listing 2.1: Timestamp Dependency vulnerabile Solidity Smart Contract code

```
1   contract Roulette {
2       uint public pastBlockTime; // forces one bet per block
3       constructor() external payable {} // initially fund contract
4       // fallback function used to make a bet
5       function () external payable {
6           require(msg.value == 10 ether); // must send 10 ether to play
7           require(now != pastBlockTime); // only 1 transaction per block
8           pastBlockTime = now;
9           if(now % 15 == 0) { // winner
10              msg.sender.transfer(this.balance);
11          }
12      }
13  }
```

2.3.5 Reentrancy

Reentrancy is a vulnerability that occurs when a smart contract calls external contracts. Most blockchain platforms that implements smart contracts provides some form of external contract calls.

Code listing 2.2: Reentrancy vulnerabile Solidity Smart Contract code

```
1    function withdraw() external {
2        uint256 amount = balances[msg.sender];
3        require(msg.sender.call.value(amount)());
4        balances[msg.sender] = 0;
5    }
```

2.3.6 Unfair Contracts

2.3.7 Scams

Chapter 3

Related work

There have been a number of literature- surveys and reviews related to Smart Contracts vulnerability detection.

Process the following literature reviews and surveys:
WOS:000473770600001 - A Survey on Security Verification of Blockchain Smart Contracts
WOS:000497160500098 - Smart Contract Security: A Software Lifecycle Perspective
WOS:000569172900008 - Verification of smart contracts: A survey
WOS:000591303900040 - Smart Contract Vulnerability Analysis and Security Audit
WOS:000618555900005 - A systematic literature review of blockchain and smart contract development: Techniques,tools, and open challenges
WOS:000630446300014 - Analysis of Blockchain Smart Contracts: Techniques and Insights
WOS:000678340800019 - Security Challenges and Opportunities for Smart Contracts in Internet of Things: A Survey
WOS:000685939000001 - Security enhancement technologies for smart contracts in the blockchain: A survey
WOS:000501392500033 - Blockchain smart contracts formalization: Approaches and challenges to address vulnerabilities

THIS IS SECONDARY REFERENCES.!!!

Sousa Matsumura *et al.* [1] conducts a systematic literature mapping. They identify several tools for analyzing SC and how to deal with the identified vulnerabilities. However, solutions for the detected vulnerabilities are primarily for avoiding the security issues, and not for correcting them.

https://ieeexplore.ieee.org/document/8689026 https://onlinelibrary.wiley.com/doi/10.1002/ett.434

3.1 General security frameworks

Lewis-Pye and Roughgarden [2] conducts a systematic literature mapping identifying

Chapter 4

Research Method

4.1 Research Motivation

Most prior works related to the detection and mitigation of software vulnerabilities have been focused on the software development life cycle. The software development process is a complex and iterative process. However, SC requires a very different approach. Due to the immutable properties of SC, all bugs and vulnerabilities needs to be removed before the code is put in production.

The area of cross-chain vulnerability analysis and detection has received limited attention. There are no papers primarily focused on cross-chain vulnerability analysis and detection. The research community is still in the early stages of the development of such a research topic. The need for a clear and systematic literature review of the current SC vulnerability detection tools and methods that may be applicable for cross-chain is prominent. Through this Systematic Literature Review (SLR), an attempt to address this will be made by answering the research questions defined in Section 4.2.

4.2 Research Questions

The research questions addressed in this SLR are:

RQ1. What are the current approaches for Smart Contract vulnerability detection?

RQ2. What is the current research on cross-chain Smart Contract vulnerability detection?

RQ3. How to make Smart Contract vulnerability detecting possible cross-chain?

4.3 Research Method and Design

The research method and design principles adopted by this SLR are based on the guidelines described by Kitchenham and Charters [3]. The process is divided into three phases.

1. Identify the need for the review, prepare a proposal for the review, and develop the review protocol.
2. Identify the research, select the studies, assess the quality, take notes and extract data, synthesize the data.
3. search and selection of studies, extracting and synthesizing the data.
4. Report the results of the review.

4.3.1 Locating Studies

In the first step, we need to locate the studies that are related to Smart Contract vulnerability detection. Web of Science (WoS) was used as the main scientific database. The following search string for WoS was defined for this literature review:

> TS=(("smart contract" OR chaincode) AND (vulnerability OR bug) AND (detection OR analysis OR tool))

Since we are also interested in the research of cross-chain Smart Contract vulnerability detection, we include "chaincode" in the search string, as this is the equivalent of "smart contract" in Hyperledger Fabric [4].

4.3.2 Study Selection and evaluation

In order to assess the retrieved literature for it's eligibility, the inclusion and exclusion criteria listed in Table 4.1 were used. No time restrictions was set.

4.3.3 Analysis and Synthesis

The search resulted in 84 publications from Web of Science (WoS). The literature was then reduced in a series of steps. This process is visualized in Figure 4.1. In the identification stage, the literature was screened for any non-english articles, missing abstracts, or duplications. During screening, 21 articles were removed based on the title, and ? were removed based on the abstract. After full-text screening at the eligibility stage, ? articles were cut. The resulting literature was reduced to a total of ??? publications.

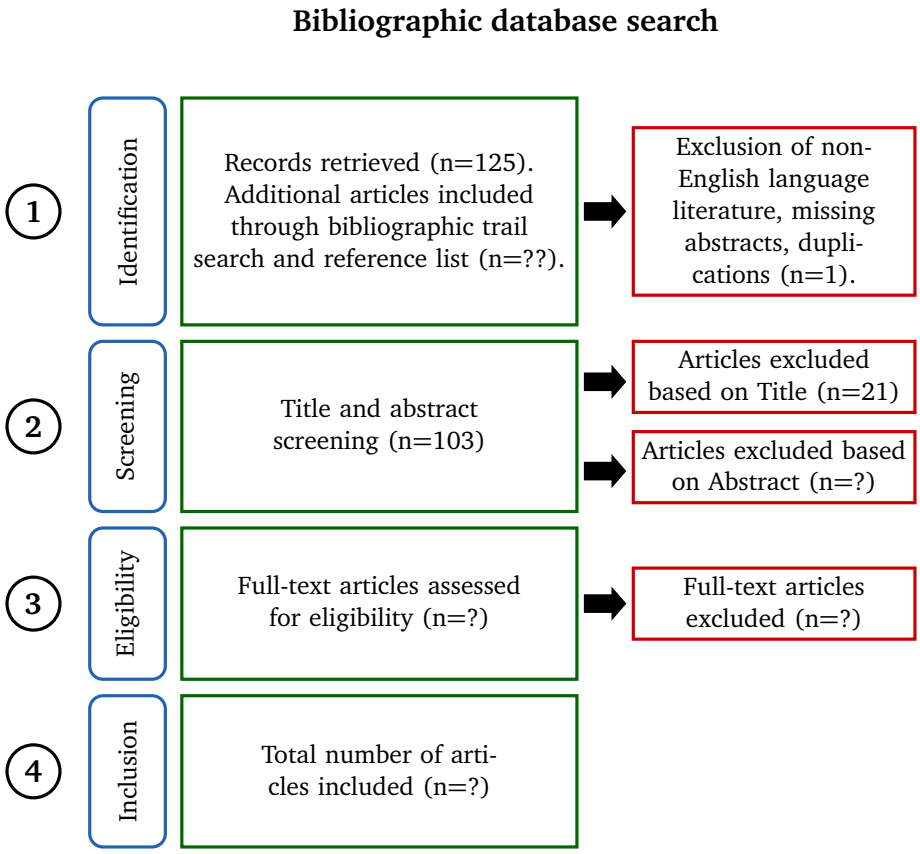

Figure 4.1: Flowchart of search and selection SLR strategy.

Table 4.1: Existing ML-based smart contract vulnerability detection tools.

	ID	Selection criteria
Inclusion	IC1	Peer-reviewed research articles, conference proceedings papers, book chapters, and serials.
	IC2	Any time-frame
Exclusion	EC1	Non english articles, missing abstracts, notes or editorials.
	EC2	Article is a copy or an older version of another publication already considered.
	EC3	Generic articles with title related to the blockchain technology and/or architecture.
	EC4	Article abstract addressing technical aspects of smart contract technology
	EC5	Article abstract addressing application (dapps) of smart contracts
	EC6	Article (full-text) not addressing vulnerability analysis or detection methods of smart contracts code.

Chapter 5

Research results

This chapter presents the results of this Systematic Literature Review (SLR). Firstly, a descriptive analysis of the selected literature is presented. Following are the results for each of the research questions defined in Section 4.2.

5.1 Descriptive analysis

This study analyzes a total of ?? research papers, published between 2018 and 2021. No publications earlier than 2018 were available. From then on, the number of publications has drastically increased. An distribution of selected literature type ordered by year is illustrated in Figure 5.1. It shows an upwards trend in term of publications. It is also to be noted that the majority of the publications are conference proceedings. This is to be expected, as the field of smart contract security is a relative new field.

5.2 Research Question 1: What are the current approaches for Smart Contract vulnerability detection?

Many tools and methods for vulnerability detection have been developed over recent years. This includes both static and dynamic vulnerability techniques, as well as tools based on Machine Learning (ML). These tools can be categorized in terms of their primary function. This includes symbolic execution, syntactical analysis, abstract interpretation, data flow analysis, fuzzy testing, formal verification and machine learning. In the following sections, the identified vulnerability detection tools are summarized, compared, and analyzed in detail (up to December 2021).

Static vulnerability detection methods are performed without actually executing programs. The analysis is performed by examining the source code or bytecode of the smart contract. Another group of analysis is so-called Dynamic program analysis. This is a form of analysis performed by executing programs on a real or virtual processor. This is in contrast to static program analysis. In order to detect

Grap hover distribution of tools of type... symbolic, ml, static, dynamic, etc...

Add snow-

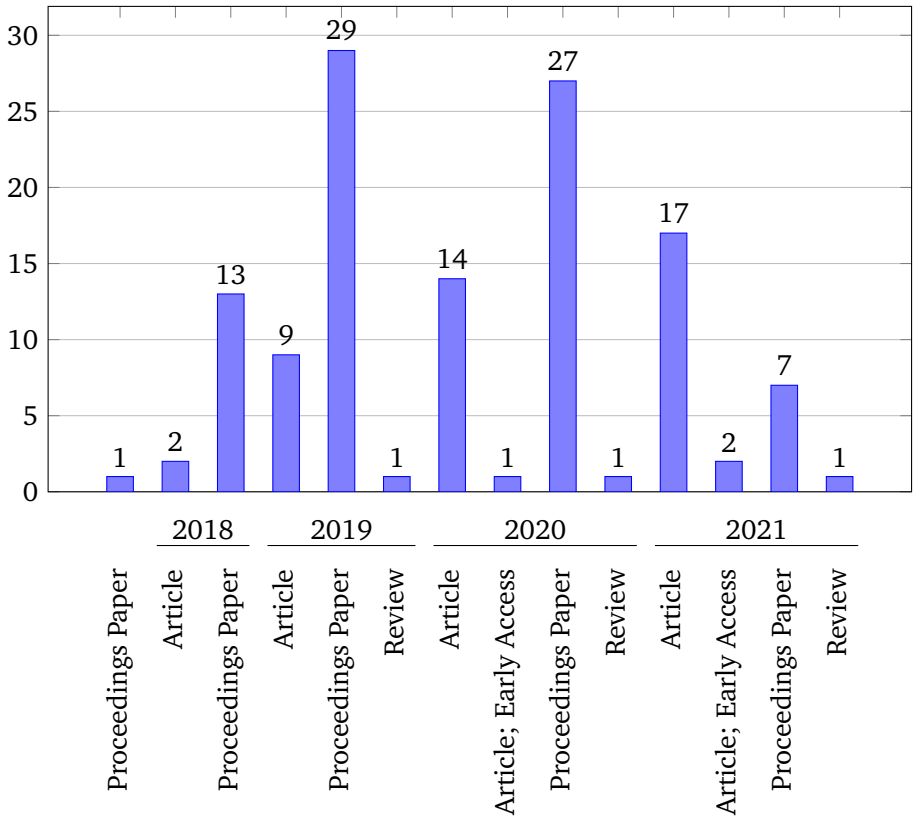

Figure 5.1: Distribution of selected literature type by year.

vulnerabilities, the program is monitored during execution. However, in order to be effective, sufficient inputs needs to be provided/tested.

What is it the other tools have done differently? Main differences, etc... Include a view on WHICH vulnerabilities can be detected by which tools.

keep 1.2textwidth (overfull margin)?

Write about yamashita2019potential - Hyperledger Fabric

Include HFContract-Fuzzer?

Include Seraph?

Table 5.1: Existing static smart contract vulnerability detection tools.

Refs.	Year	Name	Assistive technology	Capability	Input[a]
Symbolic execution					
[5]	2016	OYENTE	–	Multi-class	Solidity*, EVM bytecode
[6]	2018	ETHIR	–	General-purpose (depends on high-level verifier)	Solidity*, EVM bytecode
[7]	2018	SASC	Topological analysis and syntax analysis	Multi-class	Solidity
[8]	2018	Mythril	Taint analysis and symbolic model checking	Multi-class	EVM bytecode
[9]	2018	MPro	Taint analysis, symbolic model checking and data-flow analysis	Multi-class	EVM bytecode
[10]	2018	Maian	Concrete validation	Multi-class, dynamic	Solidity, EVM bytecode
[11]	2018	SECURIFY	Abstract interpretation	Binary decision	Solidity*, EVM bytecode
[12]	2019	SAFEVM	Symbolic model checking	General-purpose (depends on C verifier)	Solidity*, EVM bytecode
[13]	2019	SolAnalyser	Code instrumentation and execution trace analysis	Multi-class	Solidity
[14]	2019	Manticore	–		EVM bytecode
[15]	2020	RA	–	Reentrancy	EVM bytecode
Syntax analysis					
[16]	2018	SmartCheck	Topological analysis	Multi-class	Solidity
[17]	2019	NeuCheck	–	Multi-class	Solidity
[18]	2020	ReJection	–	Reentrancy	Solidity
[19]	2019	–	–	Multi-class	Go chaincode (Hyperledger Fabric)

(*Continued on next page*)

Table 5.1: (*Continued*) Existing static smart contract vulnerability detection tools.

Refs.	Year	Name	Assistive technology	Capability	Input[a]
Abstract interpretation					
[20]	2018	Zeus	Symbolic model checking		Solidity, Go, (Java, Python, JavaScript, etc.)
[21]	2018	Vandal	Logic-driven analysis	Multi-class	Bytecode
[22]	2018	MadMax	Decompilation	Multi-class gas-related	EVM bytecode
Data Flow Analysis					
[23]	2019	Slither	Taint analysis	Multi-class	Solidity
[24]	2020	CLAIRVOYANCE	Taint analysis	Reentrancy	Solidity
[25]	2020	SESCon	Taint analysis and Syntax analysis	Multi-class	Solidity
Fuzzing					
[26]	2018	ContractFuzzer	–	??	EVM ABI, EVM bytecode
[27]	2018	ReGuard	–	??	Solidity, EVM bytecode
[28]	2019	ILF	Symbolic execution	??	EVM bytecode
[29]	2020	Echidna	–	??	Solidity, Vyper [1]
[30]	2020	sFuzz	–	??	EVM bytecode

[a] Source code input that is just compiled down to bytecode is marked with "*".

5.2.1 Symbolic Execution

Symbolic execution is a ...

OYENTE. OYENTE is one of the earliest Ethereum SC vulnerability detection tool, developed by Luu *et al.* [31] in 2016. The tool consists of four main components, named CFGBuilder, Explorer, CoreAnalysis, and Validator. The CFGBuilder component builds a Control Flow Graph (CFG) of a smart contract. The Explorer component is a symbolic execution engine that explores the CFG, using the Z3 constraint solver for determining if a branch condition is either provably true or provably false along the current path. The CoreAnalysis component analyzes the explored CFG to detect vulnerabilities. The Validator component is finally used for eliminating false positives. OYENTE uses EVM bytecode as its input. OYENTE has paved the way for a lot of subsequent research. It is able to detect Call Stack Risk, Reentrancy Risk, Transaction Order Risk and Timestamp Risk.

[1] Pythonic Smart Contract Language for the EVM `https://vyper.readthedocs.io/en/stable/`

ETHIR. Albert *et al.* [6] analyzes EVM bytecode based on the rule-based representations of CFG produced by an improved version of OYENTE [31]. The improvement primarily consist of including all possible jump addresses, whereas originally OYENTE only stores the last jump address. This allows ETHIR to produce a complete CFG, containing both control-flow and data-flow information of the SC EVM bytecode. From this, ETHIR generates guarded rules for checking for conditional and unconditional jump instructions. This enables the application of (existing) high-level analyses to infer properties of the EVM code. Albert *et al.* [6] uses the high-level static analyzer SACO (Static Analyzer for Concurrent Objects) for checking conditional and unconditional jump instructions.

SAFEVM. SAFEVM is an verification tool for Ethereum smart contracts that makes use of existing verification engines for C programs. Albert *et al.* [12] uses OYENTE and ETHIR in order to produce a rule-based representation of EVM bytecode or Solidity. This is then converted into a special C program. Existing analysis tools are then used to verify the security of the converted C program, and a report with the verification results is outputted.

SASC. SASC [7] analyzes Ethereum SCs written in Solidity, and is able to detect the same logical vulnerabilities as that of OYENTE. SASC primarily makes use of symbolic execution in order to detect vulnerabilities. However, the tool also makes use of syntax analysis, combined with topological analysis, in order to locate a detected risk to a specific function.

SolAnalyser. SolAnalyser purposed by Akca *et al.* [13] uses code instrumentation and execution trace analysis in order to detect vulnerabilities in Solidity SCs. The authors proposes a fully automated pipeline for detecting vulnerabilities, as well as evaluating the tool with the help of creating a SC mutation tool.

Is this symbolic execution, or trace analysis?

Mythril. Mythril, developed by Mueller [8] is a command line tool, combining symbolic execution, with taint analysis and SMT. Mythril takes in EVM bytecode as input. In addition, Mythril also provides interfaces to allow for developers to write custom vulnerability detection logic. In order to cover the situation where a contract is called upon multiple times, Mythril simulates this through conducting multiple symbolic executions (two by default).

MPro. MPro by Zhang *et al.* [9] is a fully automated and scalable security scanning tool for Ethereum SC. MPro combines the existing tools Mythril [8] and Slither [23], leveraging both static and symbolic analysis to prune unnecessary execution paths and achieve n times efficiency increase than that of Mythril.

MAIAN. MAIAN [10] is a symbolic execution tool for specifying and inferring trace vulnerabilities. Nikolić *et al.* [10] points out that most existing tools ignore

problems related to calling a contract multiple times. Contracts containing trace vulnerabilities could: lock funds indefinitely; allow for transfer founds to any address; be killed by anyone. Based on these attributes, MAIAN marks vulnerable contracts in three categories, greedy, prodigal, and suicidal. MAIAN takes EVM bytecode as input. Along with user-defined analysis targets, this allows for confirming the presence of a trace vulnerability.

SECURIFY. SECURIFY[11] is a security analysis tool for Ethereum SC that combines abstract interpretation with symbolic execution. Thee tool automatically classifies behaviors of a contract into three categories, compliance (matched by compliance properties), violation (matched by violation properties), and warning (matched by neither). SECURIFY takes in EVM bytecode as input, along with a security model defined with a new domain-specific language. The use of a domain-specific language, enables users to express new vulnerability patterns as they emerge.

Securify has a new version named Securify2

Manticore. Manticore [14] is a flexible security analysis tool. It is based on symbolic execution and satisfiability modulo theories. It provides comprehensive API access, allowing the user to build custom symbolic execution-based tools. Further, Manticore's symbolic engine logic is decoupled from the details of a particular execution environment. This allows for support of various execution environments, such as both traditional environments(e.g., x86, ARM, and WASM), as well as Ethereum.

Investigate the effort neeeded to implement an other execution environment.

RA. RA by Chinen *et al.* [15] uses symbolic execution in order to detect re-entrancy vulnerabilities. The authors identifies that existing literature only report only program behavior via CFGs obtained within a single contract. Hence, RA focuses on analyzing re-entrancy attacks including inter-contract behavior. RA can analyze the vulnerabilities precisely, without false positives and false negatives.

5.2.2 Syntactical analysis

Syntactical analysis is a method to analyze computer programs by parsing the source code into a tree structure. This tree is then analyzed for its relation of each component.???

SmartCheck. SmartCheck is an extensible analysis tool by Tikhomirov *et al.* [16] based on syntax analysis. SmartCheck takes Solidity source code as input and translates it into an XML parse tree as an IR. This IR is then checked against XPath patterns in order to detect coding issues. The authors classifies Solidity code issues into four categories, Security, Functional, Operational and Developmental. For example, SmartCheck is able to detect Solidity specific coding issues such as style guide violations, as well as the more common blockchain security vulnerabilities like Reentrancy problems.

Findsomee way to formulate this...

NeuCheck. Lu *et al.* [17] introduces NeuCheck, a cross-platform SC syntax analysis tool for Ethereum. NeuCheck generates a syntax tree from Solidity source code, and generates an XML-based IR. The open-source tool dom4j [2] is then leveraged for parsing this tree and completing the analysis.

ReJection. Ma *et al.* [18] purpose a method for detecting SC reentrancy vulnerability detection based on analysis of ASTs. The analysis conducted by ReJection is comprised of four steps. An AST is generated from Solidity source code with the SC compiler solc [3]. This AST is then pruned for redundant information. ReJection then traverses the AST, analyzing and recording key information related to conditions of a reentrancy vulnerability. Finally, the tool decides whether there actually exists a reentrancy vulnerability, depending on some determination rules summarized by the authors.

5.2.3 Abstract interpretation

Abstract interpretation is a method to analyze computer programs by interpreting the source code as a set of logical expressions. This set of logical expressions is then analyzed for its relation of each component.

Rewrite this info.

Zeus. Zeus purposed by Kalra *et al.* [20] combines both abstract interpretation, symbolic model checking and constrained horn clauses. Model checking is verification method where a a system is modeled into a finite state machine. This is then used for verifying whether the system meets certain criteria. Zeus takes Solidity SC code as input, and translates it into an low-level IR called LLVM-IR. LLVM is a compiler toolchain, that supports a large ecosystem of code analysis tools. Along with the LLVM code, Zeus also requires a user to provide a set of policies that are used in order to .... Finally, existing LLVM Verification based tools are used on the constrained horn clauses to identify vulnerabilities.

refs..

include gray litterature - Zeus!!!

MadMax. MadMax is a gas-focused vulnerability detection tool by Grech *et al.* [22]. The authors creates a pipeline consisting of a control flow analysis based decompiler that disassembles EVM bytecode into a structured low-level IR. This is then analyzed in DataLog [32] by representing the IR as DataLog [32] facts. Along with data flow analysis along with context-sensitive flow analysis and memory layout modeling, the authors are able to automatically detect out-of-gas vulnerabilities.

EtherTrust. Grishchenko *et al.* [33]

EtherTrust

[2] `https://dom4j.github.io`

[3] `https://github.com/ethereum/solidity`

5.2.4 Data Flow Analysis

Data flow analysis is a method to analyze computer programs by analyzing the flow of data through the source code. Often taint analysis... Can analyze large programs, compared to, for example, symbolic execution.

Rewrite this info.

Slither. Slither [23] is a highly scalable static analysis tool which analyzes a smart contract source code at the intermediate representation SlithIR level. The IR is constructed on the basis of a Control Flow Graph. Slither then applies both data-flow analysis and taint analysis techniques in order to detect vulnerabilities.

CLAIRVOYANCE. CLAIRVOYANCE presents a static reentrancy detection approach [24]. The tool models cross-function and cross-contract behaviors, in order to enable a more sound analysis than Slither [23], OYENTE [31] and SECURIFY [11]. CLAIRVOYANCE applies cross-contract static taint analysis to find reentrancy candidates, then integrates path protective techniques to refine the results. The authors claim that the tool significantly outperforms the three static tools in terms of precision and accuracy.

SESCon. Ali *et al.* [25] presents a solution to detect SC vulnerabilities through static analysis. SESCon is based on XPath and taint analysis. The tool first generates a AST based on Solidity code, and applies XPath queries in order to find simple vulnerability patterns. Then, a deeper analysis is conducted based on taint analysis. For this analysis, to generate vulnerability patterns, the authors extract state variables, local variables, control flow, graph dependency of functions, and payable and non-payable functions. Ali *et al.* [25] claims that SESCon outperforms other analyzers and can accurately detect up to 90% of known vulnerability patterns.

Conclusion... Symbolic execution is the most popular vulnerability detection method. It also seems that it is the most reliable method. Most mature.

5.2.5 Fuzzing test

Fuzzing is ....

ContractFuzzer. JIANG2018CONTRACTFUZZER is a fuzzing tool for detecting several types of vulnerabilities in SCs. ContractFuzzer generates random inputs according to the Application Binary Interface (ABI) of the SC to test. It then executes the contract with these inputs and records the results. ContractFuzzer defines a set of predefined test oracles that describes specific vulnerabilities. These are used to perform the security analysis, and detect potential vulnerabilities. The authors evaluates thee tool and reports that it has lower false-positive rate than OYENTE. However, due to the randomness of the inputs, only limited system behavior is possible to cover.

ReGuard. ReGuard by Liu *et al.* [27] is another fuzzing tool, able to detect reentrancy vulnerabilities. ReGuard accepts both Solidity source code and EVM bytecode as input. It converts the input into a C++ program via an IR. This IR is an AST if the input is Solidity code, and CFG if it takes in EVM bytecode. ReGuard then generates random inputs and performs the fuzzing. ReGuard records the execution traces and feeds them into a reentrancy automata. Finally, a detection report is generated, identifying the location of the vulnerable code, along with an attack transaction that triggers the bug.

ILF. Imitation Learning based Fuzzer (ILF) [28] combines fuzzing testing with symbolic execution, through the use of imitation learning. By applying an appropriate neural network, ILF is able to learn a fuzzing probabilistic strategy, thereby imitating the behavior of symbolic execution. This way, the fuzzer is able to achieve better coverage, and thus, more vulnerabilities.

Echidna. Echidna is a SC fuzzing tool purposed by Grieco *et al.* [29], supporting user-defined analysis. Echidna consists of two main components: pre-processing and fuzzing campaign. First, the static analysis framework Slither [23] is used to extract various information. Then it applies fuzzing based on the ABI to detect violations in custom user-defined properties and assertions. Echidna is able to test both Solidity and Vyper [4] SCs.

sFuzz. Inspired by American Fuzzy Lop (AFL) [5], Nguyen *et al.* [30] created an adaptive fuzzing tool for Ethereum SCs. It also employs an efficient and lightweight adaptive strategy for selecting seeds. This is because branches guarded with strict conditions are expensive to cover. The authors report a speedup of more than two orders of magnitude compared to ContractFuzzer.

Validation

ContractLarva (runtime verification) Maian ("combines symbolic analysis and concrete validation to inspect the smart contract's bytecode. In concrete validation, the contract is executed on a fork of Ethereum for tracing and vali- dation.") Sereum

5.2.6 Formal Verification

Formal Verification is a method to mathematically verify the correctness of a program.

Rewisit this info.

[4] Pythonic Smart Contract Language for the EVM `https://vyper.readthedocs.io/en/stable/`
[5] Famous C program fuzzer

5.2.7 Machine Learning for Vulnerability Detection

"Several works have attempted to perform automated contract scan- ning using machine learning techniques. We discuss their working mechanisms and limitations below."

Table 5.2: Existing ML-based smart contract vulnerability detection tools.

Refs.	Year	Desc.[a]	Method	Feature engineering	Capability	Input
[**MISSING**]	**MISSING**	ESCORT	LSTM, transfer learning		Multi-label	Bytecode
[**MISSING**]	**MISSING**	Towards Sequential	LSTM		Binary decision	Opcode
MISSING[34]	2018	Color-inspried	CNN	??	Multi-label	Bytecode
[35]	2019	–	AST	Multi binary decision	Solidity	
[36]		Slicing matrix	??	EVM bytecode		
[37]	2020	–	Att-BLSTM	Contract snippet, word2vec	Reentrancy	Solidity
MISSING[38]	2020	NLP-inspried	AWD-LSTM	??	Multi-class	Opcode
MISSING [39]	2020	Graph NN-based	GNN	??	Multi-class	Source code
[40]	2021	–	AST	??	Solidity	
[41]	2021	ContractWard	XGBoost	Bigram model	Binary decision	Opcode
[42]	2021	–	GAN	??	Reentrancy	EVM bytecode

[a] Name of the tool or framework. If no name exists, a short description or "–" is used.

Determine category of ML-based tools.

Use multi-class, or binary decision categories?

Use the input as sections?, Eg. AST, bytecode, source

Average Stochastic Gradient Descent Weighted Dropped LSTM

Convolutional Neural Network

Graph Neural Network

AWD-LSTM based

Deep Neural Network (DNN)

VSCL. VSCL, purposed by Mi *et al.* [43], is a framework for ... VSCL accepts EVM bytecode as input, and disassembles this into opcodes. A CFG is constructed

for allowing the model to understand program runtime behavior. Further, n-gram and Term Frequency–Inverse Document Frequency (TFIDF) technique is used for generating numeric values (vectors) for features of SCs. Finally, the generated feature matrix is used as input of the DNN model. A real-world dataset is collected and labeled with the help of three tools, Oyente [31], Mythril [8], and Vandal [21].

Generative Adversarial Network (GAN)

Zhao *et al.* [42] propose a reentrant vulnerability detection model based on word embedding, similarity detection, and Generative Adversarial Networks (GANs). The model consists of six phases. Firstly, text input and semantic analysis is conducted. Then code embedding is done in step two utilizing FastText [44] for vectorization. Thirdly, contract statement matrix and vulnerability statement matrix are generated. Then a detailed detection is completed, enabling location of the actual line of vulnerable code. The discriminator is generated in the next step, finally followed by building the generator. Through this method, the authors solves the shortcomings of traditional manual data collection and manual marking. The authors report to achieve 92% detecting accuracy for reentrant vulnerable contracts.

Long Short-Term Memory (LSTM)

ESCORT. ESCORT [38] utilizes deep learning in order to detect multiple SC vulnerabilities. The model is based on a Long Short-Term Memory (LSTM) network. This model is trained on SC bytecode from the Ethereum blockchain. A tool the authors name ContractScraper is used to extract the opcodes from the SC. Further, ESCORT supports lightweight transfer learning on unseen security vulnerabilities, thus it is extensible and generalizable. Evaluation of ESCORT indicates an average accuracy of 95% (F1 score) across various vulnerability classes.

Attention-Based Bidirectional Long Short-Term Memory (Att-BLSTM)

Qian *et al.* [37] attempt to utilize a deep learning-based approach based on Attention-Based Bidirectional Long Short-Term Memory (Att-BLSTM) for detecting reentrancy bugs. The authors also purpose an contract snippet representations for SCs, intended for capturing essential semantic information and control flow dependencies specifically related to reentrancy problems. These snippets are then converted to vectors through tokenization and with the help of and word2vec [45]. the authors report good experimental results for detecting reentrancy vulnerabilities.

Bigram based

ContractWard. ContractWard [41] implements a smart contract vulnerability detection tool based on machine learning. It is a multi-label classifier that can

detect multiple vulnerabilities. Wang *et al.* [41] employs a method based on extracting bigram features from simplified opcodes. ContractWard accepts bytecode of the SC as input, and outputs a binary decision. The authors targets six vulnerabilities and employ three supervised ensemble classification algorithms, namely, XGBoost, AdaBoost and RF, and two simple classification algorithms, namely, SVM SVM and KNN. XGBoost is selected as the best performing classifier algorithm.

Abstract Syntax Tree based

How to structure in subcategories? (no name)

TODO NAME Momeni *et al.* [35] presents a machine learning predictive model that detects patterns of security vulnerabilities in SCs. The authors trained several commonly used supervised binary classifiers. This includes including Support Vector Machine (SVM), Neural Network (NN), Random Forest (RF) and Decision Tree (DT). For creating the dataset, more than 1000 SCs were collected from Etherscan [6]. For each of the contracts, an AST were generated. The authors then extract 17 features from the ASTs. For labeling the data, the existing static analysis tools Mythril [8] and Slither [23] were used. The authors report that the investigated security vulnerabilities were independent to each other. Hence, each of the classifiers were trained for each vulnerability. The evaluation results reports an average accuracy of 95% for the various vulnerabilities.

The approach described in this paper can be applied to other languages and blockchain platforms

TODO NAME Xu *et al.* [40] provides an machine learning approach to detect vulnerabilities in SCs based on Abstract Syntax Trees. The authors first generate ASTs for the contracts to analyze. They then get the ASTs of some malicious contracts. A feature vector is then generated based on common child nodes of the SCs ASTs. These vectors are then labeled with the help of existing vulnerability analysis tools, such as Slither [23] and Ethainter [**TODO**]. Machine learning classification algorithms such as KNN and SGD are then used to train a model.

Slicing matrix

Xing *et al.* [36] points out the importance of local code vulnerability. To tackle this, the authors propose a new feature extraction method named slicing matrix. The size of the feature matrix is dependent on the number of contract segments, as well as the number of different opcodes found in the dataset. Experiment results show that slice matrix improves the accuracy of vulnerable contract identification. However, the authors stress the need for better integration/use of the slice matrix in their best performing classification algorithm, namely Random Forest.

[6] `https://etherscan.io`

5.3 Research Question 2: What is the current research on cross-chain Smart Contract vulnerability detection?

"Cross-chain vulnerability detection is a method for detecting vulnerabilities in smart contract code across multiple blockchains. Little efforts have been made to develop a cross-chain vulnerability detection framework. We discuss the challenges and opportunities of cross-chain vulnerability detection."

Kalra *et al.* [20] provides a tool called Zeus. Zeus translates the Solidity SC language into LLVM-IR language. LLVM is a compiler toolchain, that supports a large ecosystem of code analysis tools. However, the LLVM-IR is very is an IR that that can be used to detect vulnerabilities in the SCs language.

Lewis-Pye and Roughgarden [2] conducts a systematic literature mapping identifying A General Framework for the Security Analysis of Blockchain Protocols

Besides the articles listed above, to my greatest extent, i have not been able to find any other research directly targeting cross-chain Smart Contract vulnerability detection. However, there are some interesting solutions that demonstrates some cross-chain support.

The most promising tool is Zeus by Kalra *et al.* [20] that allows for cross-chain compatibility through the use of LLVM. By using LLVM, any SC language can be translated into the LLVM language. This is also the only existing tool that also demonstrates cross-chain support through a prototypal mock implementation for Hyperledger Fabric [4]. The implementattion of Zeus is also not available.

Zeus: Emphasize on the need for policies and that it is a prototype implementation.

Another solution is the SC language SCILLA by Sergey *et al.* [46]. This is supposed to be an type of intermediate language based on communicating automata. However, there are no automatic translation tools for converting another language into SCILLA. Other similar solutions include converting Solidity into F*; a general-purpose functional programming language with effects aimed at program verification [47].

[35], [41] (ContractWard) eg.. most current ML solutions states that their approach is applicable to other languages and blockchain platforms. However, the compatibility assumes that the entire training process is repeated for the other target. Firstly, this is not currently feasible, as the share volume of smart contracts is not large enough on other platforms than Ethereum.

5.4 Research Question 3: How to make Smart Contract vulnerability detecting possible cross-chain?

Draft: Begin with using ML.. Then discuss existing solutions for AST analysis cross languages. Present research about cross language analysis. AST solutions, simplifications etc.

Sousa Matsumura *et al.* [1] suggest MLs for automatic vulnerability detection as future work. This because it is a promising vulnerability detection technique for traditional software, it doesn't rely heavy on human-defined rules and has

obtained competitive results.

for SCs vulnerability detection. This because MLs has been shown to be a promising vulnerability detection technique

since it is a promising vulnerability detection technique even for traditional software [7,15], does not rely heavily on rules defined by human experts and has obtained competitive results; research proposing solutions involving dynamic analysis or software testing, seeking to deal with the limitations of the execution environment.

Chapter 6

Discussion

Include principal findings, and (Strengths and Weaknesses and Meaning of findings) ... Discuss secondary literature

Several of the tools and

The results indicates that the research of vulnerability analysis of other blockchain platforms are lacking.

From the findings presented in Chapter, we can clearly see that the research of vulnerability analysis of other blockchain platforms are lacking. The area of blockchain is immature. It is therefore normal that the popularity of Ethereum steals most of the research resources. However, with the increasing popularity of other platforms, so does the need for security research on these chains. The few analyses that has been conducted on other SC-supporting chains than ... No tools available for other chains....???

Ethereum has mainly been restricted to the analysis of the security of the chain itself. Primarily consenceus protocolls,

Best practices for secure smart contracts. (some specific for ethereum)

It also seems that most research

Possible to detect ponzi schemes?

Explainable vulnerabilities in terms of locating the source of the vulnerability (by lines).

6.1 Threats to Validity

Selection process is subject to subjective criteria. Only one author.

Only use one database. Missing results... However, itdoes contain a lott of research from other database lack of snowballing (forward and backward) https://www.wohlin.eu/esem12a.p
==> more relevant publications to complement and to bring a more qualified analysis.

Chapter 7

Conclusion

include recommendations.

Practical implications for software development. Unanswered questions and implications for future research.

This article differs from many previous surveys in that it...

Bibliography

[1] G. de Sousa Matsumura, L. B. R. dos Santos, A. F. da Conceição, and N. L. Vijaykumar, *Vulnerabilities and open issues of smart contracts: A systematic mapping*, 2021. arXiv: 2104.12295 [cs.SE].

[2] A. Lewis-Pye and T. Roughgarden, *A general framework for the security analysis of blockchain protocols*, 2021. arXiv: 2009.09480 [cs.DC].

[3] B. A. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," English, Keele University and Durham University Joint Report, Tech. Rep. EBSE 2007-001, Jul. 2007. [Online]. Available: https://www.elsevier.com/__data/promis_misc/525444systematicreviewsguide.pdf.

[4] H. Foundation. "Hyperledger fabric." (), [Online]. Available: https://www.hyperledger.org/use/fabric (visited on 10/27/2021).

[5] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making Smart Contracts Smarter," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 23rd ACM Conference on Computer and Communications Security (CCS), Vienna, AUSTRIA, OCT 24-28, 2016, Assoc Comp Machinery; ACM Special Interest Grp Secur Audit & Control, 2016, 254–269, ISBN: 978-1-4503-4139-4. DOI: {10.1145/2976749.2978309}.

[6] E. Albert, P. Gordillo, B. Livshits, A. Rubio, and I. Sergey, "Ethir: A framework for high-level analysis of ethereum bytecode," in *Automated Technology for Verification and Analysis*, S. K. Lahiri and C. Wang, Eds., Cham: Springer International Publishing, 2018, pp. 513–520, ISBN: 978-3-030-01090-4.

[7] E. Zhou, S. Hua, B. Pi, J. Sun, Y. Nomura, K. Yamashita, and H. Kurihara, "Security Assurance for Smart Contract," English, in *2018 9TH IFIP INTERNATIONAL CONFERENCE ON NEW TECHNOLOGIES, MOBILITY AND SECURITY (NTMS)*, ser. International Conference on New Technologies Mobility and Security, 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS), Paris, FRANCE, FEB 26-28, 2018, IFIP TC6 5 Working Grp; IEEE; System X Inst Rech Technologique; LiP6; CNRS; TELECOM ParisTech; IEEE Commun Soc, 345 E 47TH ST, NEW YORK, NY 10017 USA: IEEE, 2018, ISBN: 978-1-5386-3662-6.

[8] B. Mueller. "Mythx tech: Behind the scenes of smart contract security analysis." (Apr. 2018), [Online]. Available: https://conference.hitb.org/hitbsecconf2018ams/sessions/smashing-ethereum-smart-contracts-for-fun-and-actual-profit/ (visited on 10/27/2021).

[9] W. Zhang, V. Ganesh, S. Banescu, L. Pasos, and S. Stewart, "MPro: Combining Static and Symbolic Analysis for Scalable Testing of Smart Contract," in *2019 IEEE 30TH INTERNATIONAL SYMPOSIUM ON SOFTWARE RELIABILITY ENGINEERING (ISSRE)*, Wolter, K and Schieferdecker, I and Gallina, B and Cukier, M and Natella, R and Ivaki, N and Laranjeiro, N, Ed., ser. Proceedings International Symposium on Software Reliability Engineering, 30th IEEE International Symposium on Software Reliability Engineering (ISSRE), Berlin, GERMANY, OCT 28-31, 2019, IEEE; IEEE Comp Soc; Bosch; Concordia; iRights Lab; German Testing Board e V; Verteilte Intelligente Systeme e V; IEEE Reliablil Soc, 2019, 456–462, ISBN: 978-1-7281-4982-0. DOI: {10.1109/ISSRE.2019.00052}.

[10] I. Nikolić, A. Kolluri, I. Sergey, P. Saxena, and A. Hobor, "Finding the greedy, prodigal, and suicidal contracts at scale," cited By 142, 2018, pp. 653–663. DOI: 10.1145/3274694.3274743. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85060022651&doi=10.1145%2f3274694.3274743&partnerID=40&md5=9bbfbf4caa9303d34c5e6dc974ece9d2.

[11] P. Tsankov, A. Dan, D. Drachsler-Cohen, A. Gervais, F. Bünzli, and M. Vechev, "Securify: Practical security analysis of smart contracts," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18, Toronto, Canada: Association for Computing Machinery, 2018, pp. 67–82, ISBN: 9781450356930. DOI: 10.1145/3243734.3243780. [Online]. Available: https://doi.org/10.1145/3243734.3243780.

[12] E. Albert, J. Correas, P. Gordillo, G. Roman-Diez, and A. Rubio, "SAFEVM: A Safety Verifier for Ethereum Smart Contracts," in *PROCEEDINGS OF THE 28TH ACM SIGSOFT INTERNATIONAL SYMPOSIUM ON SOFTWARE TESTING AND ANALYSIS (ISSTA '19)*, Zhang, DM and Moller, A, Ed., 28th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA), Beijing, PEOPLES R CHINA, JUL 15-19, 2019, Assoc Comp Machinery; ACM SIGSOFT; Microsoft Res; DiDi; Google; Huawei; MoocTest; Facebook; Fujitsu; Sourcebrella; UCLouvain, 2019, 386–389, ISBN: 978-1-4503-6224-5. DOI: {10.1145/3293882.3338999}.

[13] S. Akca, A. Rajan, and C. Peng, "Solanalyser: A framework for analysing and testing smart contracts," in *2019 26th Asia-Pacific Software Engineering Conference (APSEC)*, 2019, pp. 482–489. DOI: 10.1109/APSEC48747.2019.00071.

[14] M. Mossberg, F. Manzano, E. Hennenfent, A. Groce, G. Grieco, J. Feist, T. Brunson, and A. Dinaburg, "Manticore: A User-Friendly Symbolic Execution Framework for Binaries and Smart Contracts," in *34TH IEEE/ACM IN-*

TERNATIONAL CONFERENCE ON AUTOMATED SOFTWARE ENGINEERING (ASE 2019), 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), San Diego, CA, NOV 10-11, 2019, IEEE; Assoc Comp Machinery; IEEE Comp Soc; IEEE Comp Soc Tech Council Software Engn; ACM Special Interest Grp Artificial Intelligence; ACM Special Interest Grp Software Engn, 2019, 1186–1189, ISBN: 978-1-7281-2508-4. DOI: {10.1109/ASE.2019.00133}.

[15] Y. Chinen, N. Yanai, J. P. Cruz, and S. Okamura, "Ra: Hunting for re-entrancy attacks in ethereum smart contracts via static analysis," in *2020 IEEE International Conference on Blockchain (Blockchain)*, 2020, pp. 327–336. DOI: 10.1109/Blockchain50366.2020.00048.

[16] S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, R. Takhaviev, E. Marchenko, and Y. Alexandrov, "Smartcheck: Static analysis of ethereum smart contracts," in *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain*, ser. WETSEB '18, Gothenburg, Sweden: Association for Computing Machinery, 2018, pp. 9–16, ISBN: 9781450357265. DOI: 10.1145/3194113.3194115. [Online]. Available: https://doi.org/10.1145/3194113.3194115.

[17] N. Lu, B. Wang, Y. Zhang, W. Shi, and C. Esposito, "NeuCheck: A more practical Ethereum smart contract security analysis tool," *SOFTWARE-PRACTICE & EXPERIENCE*, vol. 51, no. 10, SI, 2065–2084, OCT 2019, ISSN: 0038-0644. DOI: {10.1002/spe.2745}.

[18] R. Ma, Z. Jian, G. Chen, K. Ma, and Y. Chen, "ReJection: A AST-based reentrancy vulnerability detection method," in *Communications in Computer and Information Science*, Springer Singapore, 2020, pp. 58–71. DOI: 10.1007/978-981-15-3418-8_5. [Online]. Available: https://doi.org/10.1007%2F978-981-15-3418-8_5.

[19] K. Yamashita, Y. Nomura, E. Zhou, B. Pi, and S. Jun, "Potential Risks of Hyperledger Fabric Smart Contracts," in *2019 IEEE 2ND INTERNATIONAL WORKSHOP ON BLOCKCHAIN ORIENTED SOFTWARE ENGINEERING (IWBOSE)*, Tonelli, R and Ducasse, S and Marchesi, M and Bracciali, A, Ed., 2nd IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE), Hangzhou, PEOPLES R CHINA, FEB 24, 2019, IEEE; IEEE Comp Soc, 2019, 1–10, ISBN: 978-1-7281-1807-9.

[20] S. Kalra, S. Goel, M. Dhawan, and S. Sharma, "Zeus: Analyzing safety of smart contracts," in *NDSS*, 2018.

[21] L. Brent, A. Jurisevic, M. Kong, E. Liu, F. Gauthier, V. Gramoli, R. Holz, and B. Scholz, *Vandal: A scalable security analysis framework for smart contracts*, 2018. arXiv: 1809.03981 [cs.PL].

[22] N. Grech, M. Kong, A. Jurisevic, L. Brent, B. Scholz, and Y. Smaragdakis, "Madmax: Surviving out-of-gas conditions in ethereum smart contracts," *Proc. ACM Program. Lang.*, vol. 2, no. OOPSLA, Oct. 2018. DOI: 10.1145/3276486. [Online]. Available: https://doi.org/10.1145/3276486.

[23] J. Feist, G. Grieco, and A. Groce, "Slither: A static analysis framework for smart contracts," *CoRR*, vol. abs/1908.09878, 2019. arXiv: 1908.09878. [Online]. Available: http://arxiv.org/abs/1908.09878.

[24] J. Ye, M. Ma, Y. Lin, Y. Sui, and Y. Xue, "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*, ser. ICSE '20, Seoul, South Korea: Association for Computing Machinery, 2020, pp. 274–275, ISBN: 9781450371223. DOI: 10.1145/3377812.3390908. [Online]. Available: https://doi.org/10.1145/3377812.3390908.

[25] A. Ali, Z. Ul Abideen, and K. Ullah, "SESCon: Secure Ethereum Smart Contracts by Vulnerable Patterns' Detection," *SECURITY AND COMMUNICATION NETWORKS*, vol. 2021, SEP 21 2021, ISSN: 1939-0114. DOI: {10.1155/2021/2897565}.

[26] B. Jiang, Y. Liu, and W. C. Chan, "ContractFuzzer: Fuzzing Smart Contracts for Vulnerability Detection," in *PROCEEDINGS OF THE 2018 33RD IEEE/ACM INTERNATIONAL CONFERENCE ON AUTOMTED SOFTWARE ENGINEERING (ASE' 18)*, Huchard, M and Kastner, C and Fraser, G, Ed., ser. IEEE ACM International Conference on Automated Software Engineering, 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE), Montpellier, FRANCE, SEP 03-07, 2018, IEEE; Assoc Comp Machinery; ACM SIGSOFT; ACM SIGAI; CNRS; IEEE CS; Huawei; Berger Levrault; Mobioos; Toyota InfoTechnol Ctr; Reg Occitanie; Inria; LIRMM; Univ Montpellier; Inst Mines Telecom Ecole Mines Telecom; Montpellier Univ Excellence; Investissements DAvenir, 2018, 259–269, ISBN: 978-1-4503-5937-5. DOI: {10.1145/3238147.3238177}.

[27] C. Liu, H. Liu, Z. Cao, Z. Chen, B. Chen, and B. Roscoe, "ReGuard: Finding Reentrancy Bugs in Smart Contracts," in *PROCEEDINGS 2018 IEEE/ACM 40TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING - COMPANION (ICSE-COMPANION*, ser. Proceedings of the IEEE-ACM International Conference on Software Engineering Companion, 40th ACM/IEEE International Conference on Software Engineering (ICSE), Gothenburg, SWEDEN, MAY 27-JUN 03, 2018, IEEE; Assoc Comp Machinery; IEEE Comp Soc; Microsoft Res, 2018, 65–68, ISBN: 978-1-4503-5663-3. DOI: {10.1145/3183440.3183495}.

[28] J. He, M. Balunović, N. Ambroladze, P. Tsankov, and M. Vechev, "Learning to fuzz from symbolic execution with application to smart contracts," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Com-*

munications Security, ser. CCS '19, London, United Kingdom: Association for Computing Machinery, 2019, pp. 531–548, ISBN: 9781450367479. DOI: 10.1145/3319535.3363230. [Online]. Available: https://doi.org/10.1145/3319535.3363230.

[29] G. Grieco, W. Song, A. Cygan, J. Feist, and A. Groce, "Echidna: Effective, usable, and fast fuzzing for smart contracts," in *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2020, Virtual Event, USA: Association for Computing Machinery, 2020, pp. 557–560, ISBN: 9781450380089. DOI: 10.1145/3395363.3404366. [Online]. Available: https://doi.org/10.1145/3395363.3404366.

[30] T. D. Nguyen, L. H. Pham, J. Sun, Y. Lin, and Q. T. Minh, *Sfuzz: An efficient adaptive fuzzer for solidity smart contracts*, 2020. arXiv: 2004.08563 [cs.SE].

[31] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making Smart Contracts Smarter," English, in *CCS'16: PROCEEDINGS OF THE 2016 ACM SIGSAC CONFERENCE ON COMPUTER AND COMMUNICATIONS SECURITY*, 23rd ACM Conference on Computer and Communications Security (CCS), Vienna, AUSTRIA, OCT 24-28, 2016, Assoc Comp Machinery; ACM Special Interest Grp Secur Audit & Control, 1515 BROADWAY, NEW YORK, NY 10036-9998 USA: ASSOC COMPUTING MACHINERY, 2016, 254–269, ISBN: 978-1-4503-4139-4. DOI: {10.1145/2976749.2978309}.

[32] Wikipedia, *Datalog — Wikipedia, the free encyclopedia*, http://en.wikipedia.org/w/index.php?title=Datalog&oldid=1053711548, [Online; accessed 13-November-2021], 2021.

[33] I. Grishchenko, M. Maffei, and C. Schneidewind, "Foundations and Tools for the Static Analysis of Ethereum Smart Contracts," in *COMPUTER AIDED VERIFICATION (CAV 2018), PT I*, Chockler, H and Weissenbacher, G, Ed., ser. Lecture Notes in Computer Science, 30th International Conference on Computer-Aided Verification (CAV) Held as Part of the Federated Logic Conference (FloC), Oxford, ENGLAND, JUL 14-17, 2018, vol. 10981, 2018, 51–78, ISBN: 978-3-319-96145-3; 978-3-319-96144-6. DOI: {10.1007/978-3-319-96145-3_4}.

[34] T. H.-D. Huang, *Hunting the ethereum smart contract: Color-inspired inspection of potential attacks*, 2018. arXiv: 1807.01868 [cs.CR].

[35] P. Momeni, Y. Wang, and R. Samavi, "Machine Learning Model for Smart Contracts Security Analysis," in *2019 17TH INTERNATIONAL CONFERENCE ON PRIVACY, SECURITY AND TRUST (PST)*, Ghorbani, A and Ray, I and Lashkari, AH and Zhang, J and Lu, R, Ed., ser. Annual Conference on Privacy Security and Trust-PST, 17th International Conference on Privacy, Security and Trust (PST), Fredericton, CANADA, AUG 26-28, 2019, IEEE; Atlantic Canada Opportunities Agcy; TD Bank; IEEE New Brunswick Sect;

CyberNB; Ignite Fredericton; ARMIS, 2019, 272–277, ISBN: 978-1-7281-3265-5.

[36] C. Xing, Z. Chen, L. Chen, X. Guo, Z. Zheng, and J. Li, "A new scheme of vulnerability analysis in smart contract with machine learning," *WIRELESS NETWORKS*, ISSN: 1022-0038. DOI: {10.1007/s11276-020-02379-z}.

[37] P. Qian, Z. Liu, Q. He, R. Zimmermann, and X. Wang, "Towards Automated Reentrancy Detection for Smart Contracts Based on Sequential Models," *IEEE ACCESS*, vol. 8, 19685–19695, 2020, ISSN: 2169-3536. DOI: {10.1109/ACCESS.2020.2969429}.

[38] A. K. Gogineni, S. Swayamjyoti, D. Sahoo, K. K. Sahu, and R. kishore, *Multi-class classification of vulnerabilities in smart contracts using awd-lstm, with pre-trained encoder inspired from natural language processing*, 2020. arXiv: 2004.00362 [cs.IR].

[39] Y. Zhuang, Z. Liu, P. Qian, Q. Liu, X. Wang, and Q. He, "Smart contract vulnerability detection using graph neural network," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, C. Bessiere, Ed., Main track, International Joint Conferences on Artificial Intelligence Organization, Jul. 2020, pp. 3283–3290. DOI: 10.24963/ijcai.2020/454. [Online]. Available: https://doi.org/10.24963/ijcai.2020/454.

[40] Y. Xu, G. Hu, L. You, and C. Cao, "A Novel Machine Learning-Based Analysis Model for Smart Contract Vulnerability," *SECURITY AND COMMUNICATION NETWORKS*, vol. 2021, AUG 15 2021, ISSN: 1939-0114. DOI: {10.1155/2021/5798033}.

[41] W. Wang, J. Song, G. Xu, Y. Li, H. Wang, and C. Su, "Contractward: Automated vulnerability detection models for ethereum smart contracts," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 2, pp. 1133–1144, 2021. DOI: 10.1109/TNSE.2020.2968505.

[42] H. Zhao, P. Su, Y. Wei, K. Gai, and M. Qiu, "GAN-Enabled Code Embedding for Reentrant Vulnerabilities Detection," in *KNOWLEDGE SCIENCE, ENGINEERING AND MANAGEMENT, PT III*, Qiu, H and Zhang, C and Fei, Z and Qiu, M and Kung, SY, Ed., ser. Lecture Notes in Artificial Intelligence, 14th International Conference on Knowledge Science, Engineering, and Management (KSEM), Tokyo, JAPAN, AUG 14-16, 2021, Springer LNCS; Waseda Univ; N Amer Chinese Talents Assoc; Longxiang High Tech Grp Inc, vol. 12817, 2021, 585–597, ISBN: 978-3-030-82153-1; 978-3-030-82152-4. DOI: {10.1007/978-3-030-82153-1_48}.

[43] F. Mi, Z. Wang, C. Zhao, J. Guo, F. Ahmed, and L. Khan, "VSCL: Automating Vulnerability Detection in Smart Contracts with Deep Learning," in *2021 IEEE INTERNATIONAL CONFERENCE ON BLOCKCHAIN AND CRYPTOCURRENCY (ICBC)*, 3rd IEEE International Conference on Blockchain and Cryptocurrency (IEEE ICBC), ELECTR NETWORK, MAY 03-06, 2021, IEEE; IEEE

Commun Soc; IBM; CSIRO, Data61, 2021, ISBN: 978-1-6654-3578-9. DOI: {10.1109/ICBC51069.2021.9461050}.

[44] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, *Enriching word vectors with subword information*, 2017. arXiv: 1607.04606 [cs.CL].

[45] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, *Distributed representations of words and phrases and their compositionality*, 2013. arXiv: 1310.4546 [cs.CL].

[46] I. Sergey, A. Kumar, and A. Hobor, *Scilla: A smart contract intermediate-level language*, 2018. arXiv: 1801.00687 [cs.PL].

[47] M. Research and Inria. "F*." (Oct. 2021), [Online]. Available: https://www.fstar-lang.org/#introduction (visited on 10/27/2021).