Cross-chain Smart Contract Vulnerability Detection Methods: A Systematic Literature Review

André Storhaug

Fall 2021

Master of Science in Computer Science
Supervisor: Jingyue Li - Associate Professor at NTNU

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Abstract

Vulnerability detection and security of Smart Contracts are of paramount importance because of their immutable nature. A Smart Contract is a program stored on a blockchain that runs when some predetermined conditions are met. While smart contracts have enabled a variety of applications on the blockchain, they may pose a significant security risk. Once a smart contract is deployed to a blockchain, it cannot be changed. It is therefore imperative that all bugs and errors are pruned out before deployment. With the increase in studies on Smart Contract vulnerability detection tools and methods, it is important to systematically review the state-of-the-art tools and methods. This, to classify the existing solutions, as well as identify gaps and challenges for future research. In this Systematic Literature Review (SLR), a total of 125 papers on Smart Contract vulnerability analysis and detection methods and tools were retrieved. These were then filtered based on predefined inclusion and exclusion criteria. Snowballing was then applied. A total of 40 relevant papers were selected and analyzed. The vulnerability detection tools and methods were classified into six categories: Symbolic execution, Syntax analysis, Abstract interpretation, Data flow analysis, Fuzzing test, and Machine learning. This SLR provides a broader scope than just Ethereum. Thus, the cross-chain applicability of the tools and methods were also evaluated. Cross-chain vulnerability detection is in this SLR defined as a method for detecting vulnerabilities in Smart Contract code that can be applied for multiple blockchains. The results of this study show that there are many highly accurate tools and methods available for Smart Contract (SC) vulnerability detection. Especially Machine Learning has in recent years drawn much attention from the research community. However, little effort has been invested in Smart Contract vulnerability detection on other chains.

Contents

Figures

Tables

Code Listings

Acronyms

ABI Application Binary Interface. 19, 24, 25

AST Abstract Syntax Tree. 22, 24–26, 31

Att-BLSTM Attention-Based Bidirectional Long Short-Term Memory. 26, 29

AWD-LSTM Average Stochastic Gradient Descent Weighted Dropped LSTM. 26, 29

CFG Control Flow Graph. 19, 20, 23–25, 30

CNN Convolutional Neural Network. 26, 30

DAO Decentralized Autonomous Organization. 1

dBFT delegated Byzantine Fault Tolerance. 4

DNN Deep Neural Network. 26, 30

DPoS Delegated Proof of Stake. 4, 5

DT Desision Tree. 26, 30

GAN Generative Adversarial Network. 26, 30

GNN Graph Neural Network. 26

IoT Internet of Things. 9

IR Intermediate Representation. vii, 22–24, *Glossary:* Intermediate Representation

KNN K-Nearest Neighbor. 31

LPoS Leased Proof of Stake. 5

LSTM Long Short-Term Memory. 26, 29

ML Machine Learning. i, 16, 25, 32, 34, 36

NFT Non Fungible Tokens. viii, 6, *Glossary:* Non Fungible Tokens

NLP Natural Language Processing. 29

PBFT Practical Byzantine Fault Tolerance. 4

PDG Program Dependency Graph. 21

PoA Proof of Authority. 5

PoS Proof of Stake. 4, 5

PoW Proof of Work. 4

RF Random Forest. 26, 31

SC Smart Contract. i, vi, ix, 1–3, 6–9, 12, 13, 16, 19–25, 27–34, 36, 37

SGD Stochastic Gradient Descent. 31

SLR Systematic Literature Review. i, iv, v, 1, 2, 12–16, 33, 34, 36, 37

SMT Satisfiability Modulo Theories. viii, 20, *Glossary:* Satisfiability Modulo Theories

SSG Symbolic Semantic Graphs. 21

SVM Support Vewctor Machine. 31

TFIDF Term Frequency–Inverse Document Frequency. 30

WoS Web of Science. viii, 13, 34, *Glossary:* Web of Science

Glossary

chaincode Chaincode is the Hyperledger Fabric equivalent of Smart Contracts. 13, 22, 23

deep learning Deep learning is a field of computer science, artificial intelligence, and statistics that deals with the learning of representations, particularly for purposes of machine learning and natural language processing. 36

fork A blockchain thath diverges into two potential paths is called a fork. 3

Intermediate Representation A representation for use as an intermediate step. 22–24

LLVM A compiler toolchain providing a complete infrastructure for creating compiler frontends and backends. 23, 32

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

Non Fungible Tokens A type of token that is unique. 6

oracle Entities that connect blockchains to external systems, enabling smart contracts to execute based upon inputs and outputs from the real world. 24

Satisfiability Modulo Theories The problem of determining whether a mathematical formula is satisfiable. 20

Solidity Solidity is a Smart Contract language developed for Ethereum. 23, 32

Web of Science Scientific citation database. 13

Z3 A theorem prover (SMT solver) from Microsoft Research. 19

Chapter 1

Introduction

The term Blockchain was popularized by Satoshi Nakamoto in 2008 with his publication of the article "Bitcoin: A Peer-to-Peer Electronic Cash System". A blockchain is a shared, immutable, digital ledger of transactions. Along with the launch of Bitcoin in 2009, this marks the birth of cryptocurrencies. In 2014, Ethereum extended the blockchain technology to include so-called Smart Contracts (SCs). A SC is a program stored on a blockchain that runs when some predetermined conditions are met. This technology has paved the way for numerous new applications across a multitude of disciplines, including finance, health, education, and many more.

Blockchain has revolutionized the way in which we are able to share data in a secure, transparent, and traceable manner. This also applies to SCs. Once a smart contract is deployed, it cannot be changed. Due to most blockchains' monetary and anonymous nature, they pose as a desirable target for adversaries and manipulators [1]. SCs often handle and store hundreds of millions of dollars worth of digital assets. This leads to deep concerns as to how to protect the assets from theft, fraud, and other forms of corruption. One of the most infamous blockchain attacks was the crowdfunding project Decentralized Autonomous Organization (DAO) hack. This hack exploited a simple vulnerability in the contract language of Ethereum, resulting in an economic loss worth about 60 million dollars at the time [2]. It is therefore imperative that all bugs and errors are pruned out before deployment, a difficult task.

There has been quite a lot of research devoted to vulnerability detection on the most popular blockchain platforms, mainly Ethereum, along with its smart contract language Solidity [3, 4]. However, not much focus has been devoted to other, less popular blockchains [5, 6]. As stated by Chen *et al.* [5], "The blockchain community keeps evolving very fast, and the most popular platform and language at one time can soon turn outdated.". With the increasing popularity of blockchain technology, so does the need for security research.

In order to better secure the vast SC ecosystem, one would need to find a way to detect and prevent these vulnerabilities across multiple blockchains (or cross-chain). In this SLR, cross-chain vulnerability detection is defined as a method for detecting vulnerabilities in smart contract code that can be applied for multiple

blockchains. The area of SCs has attracted much academic interest. Still, a vast amount of the research on security methods of SCs is scattered across scientific papers, online resources, and forums. There is a need to structure and systemize the research to continue evolving the field. In particular, the methods which are transferrable to other blockchains are of particular importance.

The purpose of this document is to investigate the current state-of-the-art in vulnerability detection, in particular cross-chain vulnerability detection. Challenges and solutions for cross-chain vulnerability detection methods will be discussed, along with purposed future work. 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?

Compared to other studies, this review will have a broader scope than just Ethereum by including other blockchain platforms. The specific contributions of this study are as follows:

- Overview of state-of-the-art tools for analysis and detection of smart contract vulnerabilities.
- Wider scope than just Ethereum, including other blockchain platforms.
- Overview of the current research on cross-chain vulnerability detection.
- Identification of open issues, possible solutions to mitigate these issues, and future directions to advance the state of research in the domain.

The rest of this paper is organized as follows. Chapter 2 describes the background of the project. The studies related to this literature review are commented in Chapter 3. Chapter 4 describes the methods used to detect vulnerabilities. Chapter 5 describes the results of the project, and Chapter 6 discuss the findings. Identified future work is presented in Chapter 7. Chapter 8 presents final remarks and concludes the paper.

Chapter 2

Background

This chapter introduces the necessary background information for this study. First, a brief introduction to blockchain technology is provided in Section 2.1 and then the concept of Smart Contracts (SCs) is introduced in Section 2.2. Finally, in Section 2.3, the most popular SC vulnerabilities are described.

2.1 Blockchain

A blockchain is a growing list of records that are linked together by a cryptographic hash. Each record is called a block. The blocks contain a cryptographic hash of the previous block, a timestamp, and transactional data. By time-stamping a block, this proves that the transaction data existed when the block was published in order to get into its hash. Since all blocks contains the hash of the previous block, they end up forming a chain. In order to tamper with a block in the chain, this also requires altering all subsequent blocks. Blockchains are therefore resistant to modification. The longer the chain, the more secure it is.

Typically, blockchains are managed by a peer-to-peer network, resulting in a publicly distributed ledger. The network is composed of nodes that are connected to each other. The nodes collectively adhere to a protocol in order to communicate and validate new blocks. Blockchain records are possible to alter through a fork. However, blockchains can be considered secure by design and presents a distributed computing system with high Byzantine fault tolerance [7].

The blockchain technology was popularized by Bitcoin in 2008. Satoshi Nakamoto introduced the formal idea of blockchain as a peer-to-peer electronic cash system. 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, Litescoin, and Ripple. Table 2.1 shows an overview of the different blockchain platforms, including the different consensus protocols, programming languages, and execution environments used. It also shows the different types of blockchains, including public, private, and hybrid. If the platform also supplies a native currency (cryptocurrency), this is also shown.

Table 2.1: Comparison of blockchain platforms.

Refs.	Platform	Consensus	Runtime env.	Smart Contract Language	Type	Cryptocurrency[a]
[8]	Ethereum	PoW and PoS	Ethereum virtual machine (EVM)	Solidity	Public	Ether
[9]	Ethereum Classic	PoW	Ethereum virtual machine (EVM)	Solidity	Public	Ether
[10]	Bitcoin	PoW	Bitcoin script engine	Bitcoin script language	Public	Bitcoin
[11]	Hyperledger Fabric	PBFT	Docker	Go, Node.js, Java, Kotlin, Python	Permissioned	–
[12]	EOS	DPoS	WASM	C++	Public	EOS
[13]	NEO	dBFT	NEO virtual machine (NeoVM)	C#, Java, Go, Python	Public	NEO
[14]	Corda	PBFT	Deterministic JVM sandbox	CorDapp Design Language (CDL)	Permissioned	–
[15]	Tezos	PoS	Interprets Michelson	Michelson	Public	Tez (XTZ)
[16]	TRON	DPoS	TRON virtual machine (TVM)	TRON Solidity	Public	Tronix (TRX).
[17]	Æternity	Hybrid PoS PoW	Aeternity EVM (AEVM) and Fast æternity Transaction Engine (FATE)	Sophia	Public	Aeternity (AE)
[18]	RChain	PoS	Rho Virtual Machine	Rholang	Hybrid	REV

(*Continued on next page*)

Table 2.1: (*Continued*) Comparison of blockchain platforms.

Refs.	Platform	Consensus	Runtime env.	Smart Contract Language	Type	Cryptocurrency[a]
[19]	Cardano	PoS	Ethereum virtual machine EVM	Plutus (Functional)	Public	–
[20]	Aergo	DPoS	AERGO Virtual Machine (AVM)	Aergo Smart Contract Language (ASCL)	Hybrid	AERGO
[21]	QTUM	PoS	Qtum x86 virtual machine	Solidity	Public	QTUM
[22]	Waves	LPoS	Interprets RIDE	RIDE	Permissioned	Waves
[23]	Vechain	PoA	Custom EVM	Solidity	Hybrid	VeChain Token (VET)

[a] Name of the cryptocurrency. If none exists "–" is used.

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. Since Ethereum's introduction of SCs, the platform has kept its market share as the most popular SC blockchain platform. Ethereum is a open, decentralized platform that allows users to create, store, and transfer digital assets. Solidity is a programming language that is used to write smart contracts in Ethereum. Solidity is compiled down to bytecode, which is then deployed and stored on the blockchain. Ethereum also introduces the concept of gas. Ethereum describes gas as follows: "It is the fuel that allows it to operate, in the same way that a car needs gasoline to run." [24]. The gas is used to pay for the cost of running the smart contract. This protects against malicious actors spamming the network [24]. The gas is paid in Wei, which is the smallest unit of Ethereum. Due to the immutable nature of blockchain technology, once a smart contract is deployed, it cannot be changed. This can have serious security implications, as vulnerable contracts can not be updated.

2.3 Smart Contract Security Vulnerabilities

There are many vulnerabilities in Smart Contracts (SCs) that can be exploited by malicious actors. Throughout the last years, an increase in the use of the Ethereum network has led to the development of SCs that are vulnerable to attacks. Due to the nature of blockchain technology, the attack surface of SCs is somewhat different from that of traditional computing systems. The Smart Contract Weakness Classification (SWC) Registry [1] collects information about various vulnerabilities. Following is a list of the most common vulnerabilities in Smart Contracts:

2.3.1 Integer Overflow and Underflow

Integer overflow and underflows happen when an arithmetic operation reaches the maximum or minimum size of a certain data type. In particular, multiplying or adding two integers may result in a value that is unexpectedly small, and subtracting from a small integer may cause a wrap to be an unexpectedly large positive value. For example, an 8-bit integer addition 255 + 2 might result in 1.

2.3.2 Transaction-Ordering Dependence

In blockchain systems, there is no guarantee on the execution order of transactions. A miner can influence the outcome of a transaction due to its own reordering

[1] `https://swcregistry.io`

criteria. For example, a transaction that is dependent on another transaction to be executed first may not be executed. This can be exploited by malicious actors.

2.3.3 Broken Access Control

Access Control issues are common in most systems, not just smart contracts. However, due to the monetary nature and openness of most SCs, properly enforcing access controls are essential. Broken access control can, for example, occur due to wrong visibility settings, giving attackers a relatively straightforward way to access contracts' private assets. However, the bypass methods are sometimes more subtle. For example, in Solidity, reckless use of `delegatecall` in proxy libraries, or the use of the deprecated `tx.origin` might result in broken access control. Code listing 2.1 shows a simple Solidity contract where anyone is able to trigger the contract's self-destruct.

Code listing 2.1: Access control vulnerable Solidity Smart Contract code

```
1 contract SimpleSuicide {
2     function sudicideAnyone() {
3         selfdestruct(msg.sender);
4     }
5 }
```

2.3.4 Timestamp Dependency

If a Smart Contract is dependent on the timestamp of a transaction, it is vulnerable to attacks. A miner has control over the execution environment for the executing SC. If the SC platform allows for SCs to use the time defined by the execution environment, this can result in a vulnerability. An example vulnerable use is a timestamp used as part of the conditions to perform a critical operation (e.g., sending ether) or as the source of entropy to generate random numbers. Hence, if the miner holds a stake in a contract, he could gain an advantage by choosing a suitable timestamp for a block he is mining. Code listing 2.2 shows an example Solidity SC code that contains this vulnerability. Here, the timestamp (the `now` keyword on line 10) is used as a source of entropy to generate a random number.

Code listing 2.2: Timestamp Dependency vulnerable Solidity Smart Contract code

```
1 contract Roulette {
2     uint public prevBlockTime; // One bet per block
3     constructor() external payable {} // Initially fund contract
4 
5     // Fallback function used to make a bet
6     function () external payable {
7         require(msg.value == 5 ether); // Require 5 ether to play
```

```
8           require(now != prevBlockTime); // Only 1 transaction per block
9           prevBlockTime = now;
10          if(now % 15 == 0) { // winner
11              msg.sender.transfer(this.balance);
12          }
13      }
14  }
```

2.3.5 Reentrancy

Reentrancy is a vulnerability that occurs when a SC calls external contracts. Most blockchain platforms that implement SC provide a way to make external contract calls. In Ethereum, an attacker may carefully construct a SC at an external address that contains malicious code in its fallback function. Then, when a contract sends funds to the address, it will invoke the malicious code. Usually, the malicious code triggers a function in the vulnerable contract, performing operations not expected by the developer. It is called "reentrancy" since the external malicious contract calls a function on the vulnerable contract and the code execution then "reenters" it. Code listing 2.3 shows a Solidity SC function where a user is able to withdraw all the user's funds from a contract. If a malicious actor carefully crafts a contract that calls the withdrawal function several times before completing, the actor would successfully withdraw more funds than the current available balance. This vulnerability could be eliminated by updating the balance (line 4) before transferring the funds (line 3).

Code listing 2.3: Reentrancy vulnerable 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  }
```

Chapter 3

Related work

This chapter provides a short overview of state-of-the-art secondary studies related to Smart Contract (SC) vulnerability analysis and detection. This is various literature that presents a survey or literature review nature.

One of the earliest studies on smart contract vulnerability analysis and detection were conducted in 2017 by Atzei *et al.* [1]. The authors provided a survey on Ethereum smart contract attacks. They analyzed the security vulnerabilities of Ethereum smart contracts and provides a taxonomy of common programming pitfalls. Later in 2018, Grishchenko *et al.* [25] published an overview of smart contract verification, covering formal semantics, security definitions, and verification tools. In the following years several surveys and reviews were conducted. Most recently in 2021, Peng *et al.* [3] published a review of known security challenges and solutions to the security issues of smart contracts in an IoT setting. Table 3.1 presents a list of some secondary studies related to smart contract vulnerability analysis and detection. All of these studies provide some insights into the current research on smart contract vulnerability analysis and detection. However, most of the research focuses exclusively on the Ethereum blockchain.

Table 3.1: Related state-of-the-art secondary studies.

Ref.	Year	Desc.	Period	Symbolic execution	Syntax analysis	Abstract interpretation	Data flow analysis	Fuzz testing	Deep learning
[1]	2017	Survey of attacks on Ethereum smart contracts	Until March 2017	●	○	○	○	○	○

(*Continued on next page*)

Table 3.1: (*Continued*) Related state-of-the-art secondary studies.

Ref.	Year	Desc.	Period	Symbolic execution	Syntax analysis	Abstract interpretation	Data flow analysis	Fuzz testing	Deep learning
[25]	2018	Overview of smart contract verification, covering formal semantics, security definitions, and verification tools	Until July 2018	●	○	●	○	○	○
[26]	2019	Overview of research on smart contract security verification, including security assurance and correctness verification	Until May 2019	○	○	○	○	●	○
[27]	2020	Study of current formalization research on all smart contract-related platforms on blockchains	From 2015 until July 2019	●	●	●	○	●	○
[28]	2019	Literature review of smart contract security from a software lifecycle perspective	Until Sept. 2019	●	●	●	○	○	○
[29]	2020	Investigate formal verification and vulnerabilities detection methods	Until June 2020	●	●	●	○	●	○
[30]	2020	Survey of the Ethereum smart contract's various vulnerabilities and the corresponding defense mechanisms	Until July 2020	●	○	○	○	○	○
[6]	2020	Survey over smart contract analysis (static analysis for vulnerability detection and program correctness, and dynamic analysis)	Until Oct. 2019	●	●	●	○	●	●
[31]	2021	Systematic literature review regarding software engineering problems and possible solutions concerning smart contracts and blockchain applications development	From 2016 until 2020	●	○	●	○	●	○

(*Continued on next page*)

Table 3.1: (*Continued*) Related state-of-the-art secondary studies.

Ref.	Year	Desc.	Period	Symbolic execution	Syntax analysis	Abstract interpretation	Data flow analysis	Fuzz testing	Deep learning
[32]	2021	Systematic mapping addressing the current state-of-art in vulnerabilities and open issues over Blockchain SCs	Until May 2021	●	●	●	●	●	●
[33]	2021	Survey of security enhancement solutions on smart contracts	Until May 2021	●	○	●	○	●	●
[3]	2021	Review of known security challenges and solutions to the security issues of smart contracts	Until April 2021	●	●	●	○	○	○

Chapter 4

Research Method

This chapter presents the research method used in this Systematic Literature Review (SLR). Firstly, the research motivation is presented, followed by the research questions defined for this SLR. Finally, the research method and design is explained.

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 blockchain technology has surged in popularity over last years, giving birth to several new blockchain platforms (see Table 2.1). As the list of new blockchains is expanding, so does the number of vulnerabilities, increasing the attack surface of the blockchain ecosystem [34]. According to Huang *et al.* [28], SC vulnerabilities may arise from the SC language used, the specific features of the blockchain, or from misunderstanding of common practices. Using tools to detect SC vulnerabilities is a key step on the road of securing SCs. However, the research area on SC security on other blockchain platforms than Ethereum (and to some degree Hyperledger Fabric) is still in its infancy. The need for a clear and systematic literature review of the current SC vulnerability detection tools and methods, regardless of platform, is therefore prominent. Further, in order to speed up the development of this research area, an investigation of tools and methods that may be applicable for cross-chain is needed. There are currently no papers primarily focused on cross-chain vulnerability analysis and detection. By making the detection approach transferable, this would accelerate the development process of vulnerability detection tools and methods. Through this SLR, an attempt to address these issues 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?

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 [35]. 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, extract and synthesize the data.
3. Report the results of the review.

4.3.1 Locating Studies

In the first step, the studies related to Smart Contract (SC) vulnerability detection were needed to locate. 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 this study is also interested in the research of cross-chain Smart Contract vulnerability detection, "chaincode" is included in the search string, as this is the equivalent of "Smart Contract" in Hyperledger Fabric [11].

4.3.2 Study Selection

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

The search resulted in 125 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. 3 parers were removed. During the screening, 21 articles were removed based on the title, and 39 were removed based on the abstract. After full-text screening at the eligibility stage, 38 articles were cut. Backwards snowballing was then applied by scanning the references of the selected

Table 4.1: Inclusion and exclusion criteria.

	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	Article not addressing vulnerability analysis or detection methods of smart contracts code.

papers. This resulted in 16 additional papers. In this SLR, the resulting literature was reduced to a total of 40 primary studies.

Figure 4.1: Flowchart of search and selection SLR strategy.

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 40 research papers published between 2016 and 2021. No publications earlier than 2016 were available. From then on, the number of publications has drastically increased and remains stable. Distribution of selected literature type ordered by year is illustrated in Figure 5.1. It shows a shift from primarily published conference papers towards articles published in journals. Still, the majority of research papers are conference articles. This is to be expected, as SC security is a relatively 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, syntax analysis, abstract interpretation, data flow analysis, fuzzy testing, and machine learning. In the following sections, the identified vulnerability detection tools are summarized, compared, and analyzed in detail.

Table 5.1 summarize the findings for the static and dynamic analysis tools. It presents the tools ordered by their primary function. In addition to the year released and the name of the tool, it also presents the assistive technology used, what inputs are required, as well as the capability of the tool. The capability is categorized as: binary decision, whether a contract is vulnerable or not; multi-class,

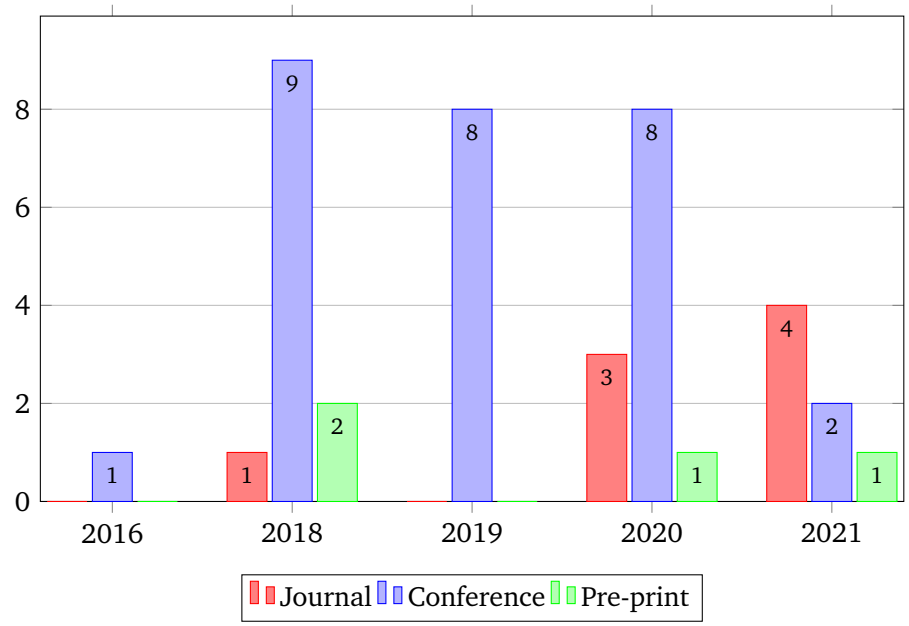

Figure 5.1: Distribution of selected literature type by year.

capable of mutual exclusively detecting several vulnerabilities; or any special vulnerability restrictions.

Table 5.1: Existing static and dynamic smart contract vulnerability detection tools.

Refs.	Year	Name[a]	Assistive technology	Capability	Input[b]
Symbolic execution					
[36]	2016	OYENTE	–	Multi-class	Solidity*, EVM bytecode
[37]	2018	ETHIR	–	General-purpose (depends on high-level verifier)	Solidity*, EVM bytecode
[38]	2018	SASC	Topological analysis and syntax analysis	Multi-class	Solidity
[39]	2018	Mythril	Taint analysis and symbolic model checking	Multi-class	EVM bytecode

(*Continued on next page*)

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

Refs.	Year	Name[a]	Assistive technology	Capability	Input[b]
[40]	2018	MPro	Taint analysis, symbolic model checking and data-flow analysis	Multi-class	EVM bytecode
[41]	2018	Maian	Concrete validation	Multi-class	Solidity, EVM bytecode
[42]	2018	SECURIFY	Abstract interpretation	Binary decision	Solidity*, EVM bytecode
[43]	2019	SAFEVM	Symbolic model checking	General-purpose (depends on C verifier)	Solidity*, EVM bytecode
[44]	2019	SolAnalyser	Code instrumentation and execution trace analysis	Multi-class	Solidity
[45]	2019	Manticore	–	Multi-class	EVM bytecode
[46]	2020	RA	–	Reentrancy	EVM bytecode
[47]	2020	Seraph	–	Multi-class	Solidity, C++, Go
Syntax analysis					
[48]	2018	SmartCheck	Topological analysis	Multi-class	Solidity
[49]	2019	NeuCheck	–	Multi-class	Solidity
[50]	2020	ReJection	–	Reentrancy	Solidity
[51]	2019	–	–	Multi-class	Go chaincode (Hyperledger Fabric)
Abstract interpretation					
[52]	2018	Zeus	Symbolic model checking	Multi-class	Solidity, Go, (Java, Python, JavaScript, etc.)
[53]	2018	Vandal	Logic-driven analysis	Multi-class	Bytecode
[54]	2018	MadMax	Decompilation	Multi-class gas-related	EVM bytecode
Data Flow Analysis					
[55]	2019	Slither	Taint analysis	Multi-class	Solidity

(*Continued on next page*)

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

Refs.	Year	Name[a]	Assistive technology	Capability	Input[b]
[56]	2020	CLAIRVOYANCE	Taint analysis	Reentrancy	Solidity
[57]	2020	Ethainter	Taint analysis	Multi-class	EVM bytecode
[58]	2020	SESCon	Taint analysis and Syntax analysis	Multi-class	Solidity
Fuzzing					
[59]	2018	ContractFuzzer	–	Multi-class	EVM ABI, EVM bytecode
[60]	2018	ReGuard	–	Multi-class	Solidity, EVM bytecode
[61]	2019	ILF	Symbolic execution	Multi-class	EVM bytecode
[62]	2020	Echidna	–	Multi-class	Solidity, Vyper [1]
[63]	2020	sFuzz	–	Multi-class	EVM bytecode

[a] Name of the tool or method. If no name exists, a short description or "–" is used.
[b] Source code input that is just compiled down to bytecode is marked with "*".

5.2.1 Symbolic execution

Symbolic execution is a method for analyzing a computer program in order to determine what inputs cause each part of a program to execute. Symbolic execution requires the program to run. During the execution of the program, symbolic values are used instead of concrete values. The program execution arrives at expressions in terms of symbols for expressions and variables, as well as constraints expressed as symbols for each possible outcome of each conditional branch of the program. Finally, the possible inputs, expressed as symbols, that trigger a branch can be determined by solving the constraints.

OYENTE. OYENTE is one of the earliest Ethereum SC vulnerability detection tool, developed by Luu *et al.* [64] 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. The tool has

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

paved the way for much subsequent research. It is able to detect Call Stack Risk, Reentrancy Risk, Transaction Order Risk, and Timestamp Risk.

ETHIR. Albert *et al.* [37] analyzes EVM bytecode based on the rule-based representations of CFG produced by an improved version of OYENTE [64]. The improvement primarily consists 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.* [37] uses the high-level static analyzer SACO (Static Analyzer for Concurrent Objects) for checking conditional and unconditional jump instructions.

SASC. SASC [38] 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.

Mythril. Mythril, developed by Mueller [39] 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 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.* [40] is a fully automated and scalable security scanning tool for Ethereum SC. MPro combines the existing tools Mythril [39], and Slither [55], leveraging both static and symbolic analysis to prune unnecessary execution paths and achieve n times efficiency increase than that of Mythril.

MAIAN. MAIAN [41] is a symbolic execution tool for specifying and inferring trace vulnerabilities. Nikolić *et al.* [41] 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 the transfer of funds 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[42] is a security analysis tool for Ethereum SC that combines abstract interpretation with symbolic execution. The 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.

SAFEVM. SAFEVM is a verification tool for Ethereum SCs that makes use of existing verification engines for C programs. Albert *et al.* [43] 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.

SolAnalyser. SolAnalyser purposed by Akca *et al.* [44] 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.

Manticore. Manticore [45] 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.

RA. RA by Chinen *et al.* [46] uses symbolic execution in order to detect reentrancy vulnerabilities. The authors identify that existing literature only reports program behavior via CFGs obtained within a single contract. Hence, RA focuses on analyzing reentrancy attacks, including inter-contract behavior. RA can analyze the vulnerabilities precisely, without false positives and false negatives.

Seraph. Yang *et al.* [47] proposed a security analysis tool called Seraph. The authors recognized that most existing tools are highly platform agnostic. They therefore designed Seraph with cross-chain support for different platforms built on EVM and WASM runtime, such as Ethereum, FISCO-BCOS, XuperChain, and EOS. Seraph accepts multiple high-level languages, for example, Solidity, C++, Go, etc. The interaction between the virtual machines and their host blockchains is achieved via a connector API. Further, inspired by Program Dependency Graphs (PDGs), the authors introduces Symbolic Semantic Graphs (SSG), a lightweight representation for security analysis.

5.2.2 Syntax analysis

Syntax analysis is a technique for analyzing computer programs by analyzing the syntactical features of a computer program. This usually involves some kind of pattern matching where the source code is first parsed into a tree structure. This tree is then analyzed by looking for vulnerable patterns while traversing the tree.

SmartCheck. SmartCheck is an extensible analysis tool by Tikhomirov *et al.* [48] 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 classify 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.

NeuCheck. Lu *et al.* [49] introduces NeuCheck, a SC syntax analysis tool for Ethereum. NeuCheck implements "cross-platform" support, meaning the tool can run on both Windows, Mac, and Linux. 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.* [50] 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.

– Yamashita *et al.* [51] introduces a tool for Hyperledger Fabric, for detecting potential vulnerabilities in chaincode. The authors use a simple approach, where the chaincode (written in Go) is converted into an AST. Several checks are applied to the AST in order to detect a total of 13 potential risks.

5.2.3 Abstract interpretation

Abstract interpretation is a method to analyze computer programs by soundly approximating the semantics of a computer program. This results in a superset of the concrete program semantics. Normally, this is then used to automatically extract information about the possible executions of computer programs.

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

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

Zeus. Zeus purposed by Kalra *et al.* [52] combines both abstract interpretation, symbolic model checking, and constrained horn clauses. Model checking is a verification method where 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 defines the correctness and/or fairness criteria for the contract. Finally, existing LLVM verification-based tools are used on the constrained horn clauses to identify vulnerabilities. As Zeus uses the LLVM toolchain, the tool can potentially be used for other blockchain platforms. The authors demonstrate this by creating a mock implementation of a Hyperledger Fabric chaincode contract.

Vandal. Vandal by Brent *et al.* [53] is a logic-driven static program analysis framework for SCs. Vandal converts EVM bytecode to semantic logic relations. Security analysis is expressed in a logic specification written in the Soufflé language. The authors reports to outperform both OYENTE, EthIR, Mythril, and Rattle analysis tools in terms of average analysis time and error rate.

MadMax. MadMax is a gas-focused vulnerability detection tool by Grech *et al.* [54]. The authors create 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 [65] by representing the IR as DataLog [65] 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.

5.2.4 Data Flow Analysis

Data flow analysis is a method for analyzing computer programs by gathering information about the flow of data through the source code. This is done by collecting all the possible set of values calculated at different points through a computer program. This method is able to analyze large programs, compared to, for example, symbolic execution.

Slither. Slither [55] 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 [56]. The tool models cross-function and cross-contract behaviors, in order to enable a more sound analysis than Slither [55], OYENTE [64] and SECURIFY [42].

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.

Ethainter. Brent *et al.* [57] presents a analysis tool named Ethainter. Ethainter performs information flow analysis in order to detect composite vulnerabilities. These are vulnerabilities that are only reached through a series of transactions. The tool accepts EVM bytecode as input. In order to leverage existing program analysis tools, primarily the Datalog language [65] and the Soufflé Datalog engine [66], the bytecode is decompiled by using GigaHorse [67]. From this, the authors are able to construct CFG with both data- and control-flow dependencies. Ethainter is deployed (together with several more analyses) for several Ethereum-based blockchains at `https://contract-library.com`

SESCon. Ali *et al.* [58] 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.* [58] claims that SESCon outperforms other analyzers and can accurately detect up to 90% of known vulnerability patterns.

5.2.5 Fuzzing test

Fuzzing is an automated testing technique for analyzing computer programs. The technique involves supplying invalid, unexpected, or random data inputs to a program in order to uncover bugs. The program is then monitored during execution for unexpected behavior such as crashes, errors, or failing built-in code assertions.

ContractFuzzer. Jiang *et al.* [59] 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 evaluate the tool and report that it has a 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.* [60] 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) [61] 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.* [62], supporting user-defined analysis. Echidna consists of two main components: pre-processing and fuzzing campaign. First, the static analysis framework Slither [55] 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.* [63] 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.

5.2.6 Machine Learning

Machine Learning (ML) is the study of computer algorithms that can automatically improve through the use of data. ML algorithms create a model based on training data. This model is then used to predict the outcome of new unseen data without being explicitly programmed to do so. Machine learning is a powerful tool for detecting vulnerabilities, as the following section shows.

Table 5.2 summarize the findings for the Machine Learning tools. It presents the year of release, the name of the tool, the ML method used, the main feature engineering method, what inputs are required, as well as the capability of the tool. The capability is categorized as: binary decision, whether a contract is vulnerable or not; multi-class, capable of mutual exclusively detecting several vulnerabilities; multi-label, capable of detecting multiple vulnerabilities; or any special vulnerability restrictions. Details about the datasets used by the Machine Learning tools are provided in Table 5.3.

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

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

Refs.	Year	Name.[a]	Method	Feature engineering	Capability	Input
[68]	2018	Color-inspried	CNN	RGB image	Multi-label	EVM Bytecode
[69]	2019	Sequential learning	LSTM	Opcode embedding	Binary decision	EVM Opcode
[70]	2019	–	AST	AST statistics	Multi binary decision	Solidity
[71]	2020	NLP-inspried	AWD-LSTM	Opcode embedding	Multi-class	EVM Opcode
[72]	2020	Graph NN-based	GNN	Normalized Graph	Multi-class	Solidity
[73]	2020	Slicing matrix	CNN, RF	# opcode occurrence	Multi-class	EVM bytecode
[74]	2020	–	Att-BLSTM	Contract snippet, word embedding	Reentrancy	Solidity
[75]	2021	–	AST	# common child nodes	Multi-class	Solidity
[76]	2021	VSCL	DNN	CFG, N-gram model	Binary decision	EVM bytecode
[77]	2021	–	GAN	Code embedding	Binary decision	EVM bytecode
[78]	2021	ESCORT	LSTM, transfer learning	Word embedding	Multi-label	EVM Bytecode
[79]	2021	ContractWard	DT	Bigram model	Binary decision	EVM Opcode

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

Table 5.3: Existing ML-based smart contract vulnerability detection tools data sets.

Refs.	Year	Name[a]	Data source	Training[b]	Testing	Labeling	Balancing	Accessibility[c]
[68]	2018	Color-inspried	Crawled Etherscan for contracts from Jan. 2018 to Apr. 2018, and from May. 2018 to Jun 2018.	–	–	–	–	○
[69]	2019	Sequential learning	Sourced 920,179 Ethereum SCs from Google BigQuery. After balancing, the dataset is reduced to 620,000 SCs.	80%	20%	Maian	SMOTE	●
[70]	2019	–	Collects 13,745 unique Solidity SCs from Etherscan. Reduced to 1,013 contracts based on Solidity compiler version 4.	80%	20%	Mythril, Slither	–	◐
[71]	2020	NLP-inspried	Dataset is obtained from the work of Tann *et al.* [69], containing 892,913 SCs labeled in five categories. Only four categories are used, resulting in 40,877 SCs.	80%	20%	Maian	Reduces imbalance by only considering distinct opcode combinations.	◐
[72]	2020	Graph NN-based	40,932 SCs from Ethereum blockchain.	20%	80%	–	–	○
[73]	2020	Slicing matrix	Collects 19,145 SCs from Ethereum.	–	–	–	–	○
[74]	2020	–	Creates the *contract snippets* dataset consisting of 2000 SCs containing reentrant markers. These are extracted from 42,000 SCs scraped from Etherscan.	80%	20%			●

(*Continued on next page*)

Table 5.3: (*Continued*) Existing ML-based smart contract vulnerability detection tools data sets.

Refs.	Year	Desc.[a]	Data source	Training[b]	Testing	Labeling	Balancing	Accessibility[c]
[75]	2021	–	Combines existing datasets: Smartbugs, SolidiFi, and Smartbugs-wild (unlabeled). Total: 408 SCs.	70%	30%	Slither, Ethainter	–	◐
[76]	2021	VSCL	All contracts deployed on Ethereum as of May 2019. Removed duplicates, resulting in 205,848 unique SCs.	80%	20%	OYENTE, Mythril, Vandal	–	◐
[77]	2021	–	70 Solidity contracts from Etherscan.	20 SCs	50 SCs	Manual	NA	◐
[78]	2021	ESCORT	93,497 SCs selected from ∼1.2 million contracts scraped from first 5 million Ethereum blocks.	80%	20%	OYENTE, Mythril, Dedaub	Equally-sized vulnerability distribution selection.	◐
[79]	2021	ContractWard	49,502 Solidity SCs from Ethereum official website.	70%	30%	OYENTE	SMOTETomek	◐

[a] Name of the tool or method. If no name exists, a short description or "–" is used.
[b] In the "Training" column, the data used for both training and validation is included.
[c] In the "Accessibility" column, the symbols ●, ◐, and ○denotes that the dataset is open to access, mostly reproducible, or irreproducible.

Long Short-Term Memory (LSTM)

Sequential learning Tann *et al.* [69] purposes a sequential based deep learning based approach in order to detect whether a SC is vulnerable or not (binary decision). The model used is a LSTM network. The input is code embedded SC bytecode. The authors report that the results suggest that sequential learning of SC security threats provides significant improvements over symbolic analysis tools. They achieve a detection accuracy of 99.57% and an F_1 score of 86.04%.

ESCORT. ESCORT [78] 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 download the SCs and extract the opcodes. 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.* [74] 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 a contract snippet representation 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 the word embedding tool word2vec [80]. The authors report good experimental results for detecting reentrancy vulnerabilities.

Average Stochastic Gradient Descent Weighted Dropped LSTM

– Gogineni *et al.* [71] presents an deep learning method, named Average Stochastic Gradient Descent Weighted Dropped LSTM (AWD-LSTM). The authors takes inspiration from Natural Language Processing (NLP) methods. In particular, they employ a method where a neural network is trained initially on a different target task for which a large amount of data is available. Then, by replacing some blocks in the initial network, the neural network architecture is modified to perform the required target task. Specifically, the authors used two neural networks where the first network learns semantic information about the input data which helps the second network to achieve better and quicker performance. Gogineni *et al.* [71] report the proposed method is fairly accurate and produces acceptable results with an accuracy of 91.0% and an F_{beta} score of 90.0%.

Convolutional Neural Network

Color-inspired. Huang [68] purpose to transform SC EVM bytecode to RGB images and feed the images through a Convolutional Neural Network (CNN). The type of classification used is binary classification. However, their method is able to support multi-label classification by retraining the obtained model on the corresponding dataset.

Graph Neural Network

– Zhuang *et al.* [72] purpose using graph neural network to classify vulnerable Ethereum SCs. the input code. The method supports multi-class detection and operates on Solidity code as input. The method consists of three phases. First is the graph generation phase, where CFG and data flow semantics are extracted from the source code. Then, the graph is normalized. Finally, a novel message propagation network for vulnerability modeling and detection is generated.

Deep Neural Network

VSCL. VSCL, purposed by Mi *et al.* [76], is a novel vulnerability detection framework for SCs. 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 [64], Mythril [39], and Vandal [53].

Generative Adversarial Network

– Zhao *et al.* [77] 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 [81] for vectorization. Thirdly, the contract statement matrix and vulnerability statement matrix are generated. Then a detailed detection is completed, enabling the 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 solve the shortcomings of traditional manual data collection and manual marking. The authors report achieving a 92% detecting accuracy for reentrant vulnerable contracts.

Desision Tree

ContractWard. ContractWard [79] implements a SC vulnerability detection tool based on machine learning. It is a multi-label classifier that can detect multiple

vulnerabilities. Wang *et al.* [79] 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 target 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.

Slicing matrix. Xing *et al.* [73] 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.

Abstract Syntax Tree

– Momeni *et al.* [70] 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 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 was generated. The authors then extract 17 features from the ASTs. For labeling the data, the existing static analysis tools Mythril [39] and Slither [55] 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.

– Xu *et al.* [75] 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 [55], and Ethainter [57]. Machine learning classification algorithms such as KNN and SGD are then used to train a model.

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

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

The number of blockchains is increasing rapidly. It is therefore important to systematize the current research on cross-chain Smart Contract vulnerability detection. To my greatest ability, I have not been able to find any research directly targeting cross-chain Smart Contract vulnerability detection. Although, there are some interesting works that have potential. This section presents the current tools and methods that demonstrate cross-chain support to some extent.

Kalra *et al.* [52] purpose the tool called Zeus. As described in Section 5.2.3, Zeus translates Solidity SC code into LLVM-IR language. The use of LLVM bytecode enables Zeus to theoretically support SC verification for different blockchain platforms. This is doable due to the separation of the translation from the implementation of the verification checks. The authors implement full support for Ethereum. They also demonstrate the generalizability through implementing crude support for Hyperledger Fabric by manually porting a Smart Contract (SC) to GO and linking it against Fabric's mock-stub. However, the implementation is closed source and ZEUS only provides an informal description of the translation to LLVM [82].

Seraph, purposed by Yang *et al.* [47] is another security analysis tool that demonstrates some cross-chain capability. Through its connector API (see Section 5.2.1), Seraph is capable of supporting multiple blockchain platforms that either run on EVM or WASM. Hence, it is able to support platforms such as Ethereum, FISCO-BCOS, XuperChain, and EOS.

Another interesting area is the use of ML. Since MLs does not rely heavily on human-defined rules and has obtained competitive results, it is a promising research area for automatic vulnerability detection. Most of the solutions described in Section 5.2.6 could probably be re-created on other blockchain platforms. Although, none of the authors have actually attempted to do so. Their approach requires that the entire training process is repeated for the other target platform. This is not currently feasible, as the share volume of smart contracts is not large enough on other platforms than Ethereum. Further, due to the difference of the various blockchain platforms, the models and feature engineering might need a lot of customization for the individual platform.

Chapter 6

Discussion

In this Systematic Literature Review (SLR), efforts are made to systemize the state-of-the-art research on various blockchain platforms. A total of 40 papers available as of 2021 has been discussed and analyzed. From the findings presented in Chapter 5, we can clearly see that the research of vulnerability analysis of other blockchain platforms than Ethereum is lacking. Even with deliberate efforts for covering other blockchain platforms, almost all research focuses exclusively on the Ethereum blockchain, with some minor exceptions being Hyperledger Fabric. The area of blockchain is immature. Therefore, it is expected that the popularity of Ethereum steals most of the research resources.

6.1 Comparison with related work

Compared to related surveys on the topic of smart contract vulnerability analysis and detection [1, 3, 6, 25–33] (see Chapter 3), this study covers a wider range of blockchain platforms. Where most surveys focus on the Ethereum blockchain, this study has deliberately focused on all SC vulnerability detection tools regardless of the platform used. The closest related work to this study is by Wang *et al.* [33]. The authors provide a survey of security enhancement solutions on smart contracts. The survey addresses most static and dynamic analysis techniques, as well as some deep learning methods. Further, they state they provide a widened scope beyond just Ethereum. Compared to [33], this study provides a more up-to-date Systematic Literature Review (SLR) until 2021. It provides a twice as long list of vulnerability analysis and detection methods, as well as a more detailed analysis of each tool and its capabilities. Further, this study also focuses on the cross-applicability of the various detection methods. A comparison of this SLR and [33] can be found in Table 6.1.

Table 6.1: Comparison of this SLR with closest related work(Wang *et al.* [33]).

	This SLR	**[33]**
Period covered	Until 2021	Until May 2021
Scientific databases searched	Web of Science	CM Digital Library IEEE Xplore Elsevier Science Direct
Cross-chain applicability	Yes	No
Ethereum specific scope	No	No
Number of tools identified	40	20[a]
Scope	SC Vulnerability Detection tools and Methods	SC Security Enhancing Solutions
Categories	Symbolic execution Syntax analysis Abstract interpretation Data flow analysis Fuzzing test Machine learning	Symbolic execution Abstract interpretation Fuzzing test Formal verification Deep learning Privacy enhancing techniques
Table comparison attributes for static and dynamic analysis	Year Main technology Assistive technology Capability Input	Main technology Assistive technology Level Type
Table comparison attributes for ML	Year Method Feature engineering Capability Input Dataset Dataset availability	NA
Considers ML datasets	Yes	No

[a] Papers classified as "Formal verification" or "Privacy enhancing techniques" are not counted as they are not vulnerability detection tools or methods.

6.2 Threats to Validity

The search strategy applied poses a likely threat to validity from missing out on or excluding relevant papers. Further, only one database was used for searching. To mitigate this, the search terms were iteratively improved. Further, an extensive snowballing process on references of the selected papers to identify related papers was conducted. During study selection, the researcher's subjective judgment could be a threat. The pre-defined review protocol was strictly followed in order to mitigate this. A widening of the search scope, as well as increasing the number of authors, could provide more relevant publications to complement and result in

a more qualified analysis.

Chapter 7

Future work

The area of SC vulnerability analysis and detection has already come a long way, even though the area of blockchain is still in its infancy. There are still many research gaps needing to be filled. From this SLR, one can see there is a noticeable lack in research on vulnerability detection for other blockchain platforms than Ethereum. Further, there is a need for more research on the analysis of what vulnerabilities apply to the various blockchain platforms.

As discussed earlier in Section 5.3, several solutions for Ethereum may be applicable for cross-chain detection. In the area of deep learning, there is significant potential in creating deep learning models that can function across different platforms. There is a need for assessing the cross-chain applicability of currently available tools. However, a significant limitation for these solutions is the lack of training data for blockchain platforms. Ethereum is currently the only platform with sufficient deployed Smart Contracts (SCs) in order to be able to fuel machine learning. Further, as can be seen from Table 5.3, almost all existing ML solutions have to create their own datasets from scratch. A sound way to produce adequate training data for other blockchain platforms would therefore be required in order to allow for deep learning models to function for multiple platforms. This would make it more desirable for other people to start using other blockchain platforms, and in turn, encourage researchers to expand their research area to other blockchain platforms.

Chapter 8

Conclusion

This paper presents the results of a Systematic Literature Review of existing Smart Contract vulnerability analysis and detection methods. The motivation for this study was to provide a state-of-the-art overview of the current situation of the SC vulnerability detection. A total of 40 primary studies were selected based on predefined inclusion and exclusion criteria. A systematic analysis and synthesis of the data were extracted from the papers, and comprehensive reviews were performed. Further, to the greatest extent, this paper also identifies the current available cross-chain tools and methods. The cross-chain applicability for these assets is investigated and analyzed.

The findings from this study show that there are a number of methods and implemented tools readily available for vulnerability analysis and detection. Several of these tools show great results. The most prevalent methods are static analysis tools, where symbolic execution is among the most popular. Other methods such as syntax analysis, abstract interpretation, data flow analysis, fuzzy testing, and machine learning are also readily used. In this paper, some potential cross-chain tools are highlighted and discussed. Although they pose several limitations, they show significant potential for further development. Especially interesting is the potential machine learning-based cross-chain detection methods.

From this study, one can see that there is a significant lack of research on vulnerability detection on other blockchain platforms than Ethereum. The hope is that the results from this study provide a starting point for future research on cross-chain analysis.

Bibliography

[1] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on ethereum smart contracts (sok)," in *International conference on principles of security and trust*, Springer, 2017, pp. 164–186.

[2] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on ethereum smart contracts (sok)," in *Principles of Security and Trust*, M. Maffei and M. Ryan, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2017, pp. 164–186, ISBN: 978-3-662-54455-6.

[3] K. Peng, M. Li, H. Huang, C. Wang, S. Wan, and K.-K. R. Choo, "Security Challenges and Opportunities for Smart Contracts in Internet of Things: A Survey," *IEEE INTERNET OF THINGS JOURNAL*, vol. 8, no. 15, 12004–12020, Aug. 2021, ISSN: 2327-4662. DOI: {10.1109/JIOT.2021.3074544}.

[4] A. Singh, R. M. Parizi, Q. Zhang, K.-K. R. Choo, and A. Dehghantanha, "Blockchain smart contracts formalization: Approaches and challenges to address vulnerabilities," *Computers & Security*, vol. 88, p. 101 654, 2020, ISSN: 0167-4048. DOI: 10.1016/j.cose.2019.101654. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404818310927.

[5] H. Chen, M. Pendleton, L. Njilla, and S. Xu, "A survey on ethereum systems security: Vulnerabilities, attacks, and defenses," *ACM Comput. Surv.*, vol. 53, no. 3, Jun. 2020, ISSN: 0360-0300. DOI: 10.1145/3391195. [Online]. Available: https://doi.org/10.1145/3391195.

[6] S. Kim and S. Ryu, "Analysis of Blockchain Smart Contracts: Techniques and Insights," in *2020 IEEE SECURE DEVELOPMENT (SECDEV 2020)*, IEEE Secure Development (IEEE SecDev) Conference, ELECTR NETWORK, SEP 28-30, 2020, IEEE Comp Soc; NSF; Cisco; Off Naval Res, 2020, 65–73, ISBN: 978-1-7281-8388-6. DOI: {10.1109/SecDev45635.2020.00026}.

[7] L. S. Sankar, M. Sindhu, and M. Sethumadhavan, "Survey of consensus protocols on blockchain applications," in *2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS)*, 2017, pp. 1–5. DOI: 10.1109/ICACCS.2017.8014672.

[8] Ethereum. "Ethereum." (Jan. 2022), [Online]. Available: https://ethereum.org/en/ (visited on 01/10/2022).

[9] Ethereum. “Ethereum classic.” (Dec. 2021), [Online]. Available: https://ethereumclassic.org (visited on 01/10/2022).

[10] B. Project. “Bitcoin.” (Jan. 2022), [Online]. Available: https://bitcoin.org/en/ (visited on 01/10/2022).

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

[12] Block.one. “Eosio.” (Jan. 2022), [Online]. Available: https://eos.io (visited on 01/10/2022).

[13] N. Team. “Neo.” (Jan. 2022), [Online]. Available: https://neo.org (visited on 01/10/2022).

[14] r3. “Corda.” (Jan. 2022), [Online]. Available: https://www.corda.net (visited on 01/10/2022).

[15] tezoss. “Tezos.” (Jan. 2022), [Online]. Available: https://www.tezos.com (visited on 01/10/2022).

[16] tron dao. “Tron.” (Jan. 2022), [Online]. Available: https://tron.network (visited on 01/10/2022).

[17] ætternity. “Ætternity.” (Jan. 2022), [Online]. Available: https://aeternity.com (visited on 01/11/2022).

[18] rchain. “Rchain.” (Jan. 2021), [Online]. Available: https://rchain.coop (visited on 01/11/2022).

[19] Cardano. “Cardano.” (Jan. 2021), [Online]. Available: https://cardano.org (visited on 01/11/2022).

[20] Aergo. “Aergo.” (Jan. 2021), [Online]. Available: https://www.aergo.io (visited on 01/11/2022).

[21] Q. C. FOUNDATION. “Qtum.” (Jan. 2021), [Online]. Available: https://www.aergo.io (visited on 01/11/2022).

[22] Waves. “Wawes.” (Jan. 2021), [Online]. Available: https://waves.tech (visited on 01/11/2022).

[23] V. Foundation. “Vechain.” (Jan. 2021), [Online]. Available: https://www.vechain.org (visited on 01/11/2022).

[24] Ethereum. “Gas and fees.” (Dec. 2021), [Online]. Available: https://ethereum.org/en/developers/docs/gas/ (visited on 01/04/2022).

[25] 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. DOI: {10.1007/978-3-319-96145-3_4}.

[26] J. Liu and Z. Liu, "A Survey on Security Verification of Blockchain Smart Contracts," *IEEE ACCESS*, vol. 7, 77894–77904, 2019, ISSN: 2169-3536. DOI: {10.1109/ACCESS.2019.2921624}.

[27] A. Singh, R. M. Parizi, Q. Zhang, K.-K. R. Choo, and A. Dehghantanha, "Blockchain smart contracts formalization: Approaches and challenges to address vulnerabilities," *COMPUTERS & SECURITY*, vol. 88, Jan. 2020, ISSN: 0167-4048. DOI: {10.1016/j.cose.2019.101654}.

[28] Y. Huang, Y. Bian, R. Li, J. L. Zhao, and P. Shi, "Smart Contract Security: A Software Lifecycle Perspective," *IEEE ACCESS*, vol. 7, 150184–150202, 2019, ISSN: 2169-3536. DOI: {10.1109/ACCESS.2019.2946988}.

[29] M. Almakhour, L. Sliman, A. E. Samhat, and A. Mellouk, "Verification of smart contracts: A survey," *PERVASIVE AND MOBILE COMPUTING*, vol. 67, Sep. 2020, ISSN: 1574-1192. DOI: {10.1016/j.pmcj.2020.101227}.

[30] D. He, Z. Deng, Y. Zhang, S. Chan, Y. Cheng, and N. Guizani, "Smart Contract Vulnerability Analysis and Security Audit," *IEEE NETWORK*, vol. 34, no. 5, 276–282, Sep. 2020, ISSN: 0890-8044. DOI: {10.1109/MNET.001.1900656}.

[31] A. Vacca, A. Di Sorbo, C. A. Visaggio, and G. Canfora, "A systematic literature review of blockchain and smart contract development: Techniques, tools, and open challenges," *JOURNAL OF SYSTEMS AND SOFTWARE*, vol. 174, Apr. 2021, ISSN: 0164-1212. DOI: {10.1016/j.jss.2020.110891}.

[32] 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].

[33] Y. Wang, J. He, N. Zhu, Y. Yi, Q. Zhang, H. Song, and R. Xue, "Security enhancement technologies for smart contracts in the blockchain: A survey," *TRANSACTIONS ON EMERGING TELECOMMUNICATIONS TECHNOLOGIES*, 2021, ISSN: 2161-3915. DOI: {10.1002/ett.4341}.

[34] M. Gogan. "Smart contract security: What are the weak spots of ethereum, eos, and neo networks?" (Dec. 2018), [Online]. Available: https://technative.io/smart-contract-security-what-are-the-weak-spots-of-ethereum-eos-and-neo-networks/ (visited on 01/17/2022).

[35] 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.

[36] 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}.

[37] 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.

[38] 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.

[39] 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).

[40] 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}.

[41] 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.

[42] 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.

[43] 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}.

[44] 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.

[45] 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 INTERNATIONAL 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}.

[46] 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.

[47] Z. Yang, H. Liu, Y. Li, H. Zheng, L. Wang, and B. Chen, "Seraph: Enabling cross-platform security analysis for evm and wasm 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. 21–24, ISBN: 9781450371223. DOI: 10.1145/3377812.3382157. [Online]. Available: https://doi.org/10.1145/3377812.3382157.

[48] S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, R. Takhaviev, E. Marchenko, and Y. Alexandrov, "Smartcheck: Static analysis of ethereum smart con-

tracts," 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.

[49] 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}.

[50] 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.

[51] 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.

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

[53] 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].

[54] 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.

[55] 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.

[56] 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.

[57] L. Brent, N. Grech, S. Lagouvardos, B. Scholz, and Y. Smaragdakis, "Ethainter: A Smart Contract Security Analyzer for Composite Vulnerabilities," in *PROCEEDINGS OF THE 41ST ACM SIGPLAN CONFERENCE ON PROGRAMMING LANGUAGE DESIGN AND IMPLEMENTATION (PLDI '20)*, Donaldson, AF and Torlak, E, Ed., 41st ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), ELECTR NETWORK, JUN 15-20, 2020, Assoc Comp Machinery; ACM SIGPLAN, 2020, 454–469, ISBN: 978-1-4503-7613-6. DOI: {10.1145/3385412.3385990}.

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

[59] 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}.

[60] 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}.

[61] 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 Communications 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.

[62] 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. IS-

STA 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.

[63] 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].

[64] 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}.

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

[66] H. Jordan, B. Scholz, and P. Subotić, "Soufflé: On synthesis of program analyzers," in *Computer Aided Verification*, S. Chaudhuri and A. Farzan, Eds., Cham: Springer International Publishing, 2016, pp. 422–430, ISBN: 978-3-319-41540-6.

[67] N. Grech, L. Brent, B. Scholz, and Y. Smaragdakis, "Gigahorse: Thorough, declarative decompilation of smart contracts," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 2019, pp. 1176–1186. DOI: 10.1109/ICSE.2019.00120.

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

[69] W. J.-W. Tann, X. J. Han, S. S. Gupta, and Y.-S. Ong, *Towards safer smart contracts: A sequence learning approach to detecting security threats*, 2019. arXiv: 1811.06632 [cs.CR].

[70] 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.

[71] 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].

[72] 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.

[73] 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*, 2020, ISSN: 1022-0038. DOI: {10.1007/s11276-020-02379-z}.

[74] 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}.

[75] 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. 2021, ISSN: 1939-0114. DOI: {10.1155/2021/5798033}.

[76] 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}.

[77] 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. DOI: {10.1007/978-3-030-82153-1_48}.

[78] O. Lutz, H. Chen, H. Fereidooni, C. Sendner, A. Dmitrienko, A. R. Sadeghi, and F. Koushanfar, *Escort: Ethereum smart contracts vulnerability detection*

using deep neural network and transfer learning, 2021. arXiv: 2103.12607 [cs.CR].

[79] 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.

[80] 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].

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

[82] Y. Wang, S. Lahiri, S. Chen, R. Pan, I. Dillig, C. Born, and I. Naseer, "Formal specification and verification of smart contracts for azure blockchain," Apr. 2019. [Online]. Available: https://www.microsoft.com/en-us/research/publication/formal-specification-and-verification-of-smart-contracts-for-azure-blockchain/.