Smart Contract Vulnerability Detection Methods: A Systematic Literature Review

André Storhaug

Fall 2021

Abstract

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

Sammendrag

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

Contents

Figures

Tables

Code Listings

Acronyms

AST Abstract Syntax Tree. 17

CFG Control Flow Graph. 14, 18

IR Intermediate Representation. viii, 17, 18, 21, *Glossary:* Intermediate Representation

ML Machine Learning. 22

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

SC Smart Contract. iv, vii, ix, 2–5, 7, 8, 11, 14, 16–18, 21–23

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

SMT Satisfiability Modulo Theories. viii, 16, *Glossary:* Satisfiability Modulo Theories

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

Glossary

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

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

F* General-purpose functional programming language with effects aimed at program verification. 22

gas Gas in crypto refers to the computational effort required to execute operations. This is normally paid in the blockchain's plattform cryptocurrency. 13

Intermediate Representation A representation for use as an intermediate step. 17, 18, 21

LLVM A compiler toolchain providing a complete infrastructure for creating compiler frontends and backends. 18, 21

LLVM-IR LLVM-IR is the intermediate representation of the LLVM compiler toolchain. 18, 21

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

Satisfiability Modulo Theories The problem of determining whether a mathematical formula is satisfiable. 16

Solidity Solidity is a Smart Contract language developed for Ethereum. 18, 21, 22

Web of Science Scientific citation database. 8

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

Todo list

Chapter 1

Introduction

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

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

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

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

call it paper or document?

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

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

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

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

Chapter 2

Background

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

2.1 Blockchain

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

2.1.1 Ethereum

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

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

2.2 Smart Contracts

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

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

2.3 Smart Contract Security Vulnerabilities

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

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

2.3.1 Integer Overflow and Underflow

2.3.2 Transaction-Ordering Dependence

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

2.3.3 Stack Size Limit

2.3.4 Timestamp Dependency

Considering changing to "Block state dependence"

Code listing 2.1: Timestamp Dependency vulnerabile Solidity Smart Contract code

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

2.3.5 Reentrancy

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

Code listing 2.2: Reentrancy vulnerabile Solidity Smart Contract code

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

2.3.6 Unfair Contracts

2.3.7 Scams

Chapter 3

Related work

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

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

THIS IS SECONDARY REFERENCES.!!!

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

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

3.1 General security frameworks

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

Chapter 4

Research Method

4.1 Research Motivation

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

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

4.2 Research Questions

The research questions addressed in this SLR are:

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

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

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

4.3 Research Method and Design

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

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

4.3.1 Locating Studies

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

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

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

4.3.2 Study Selection and evaluation

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

4.3.3 Analysis and Synthesis

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

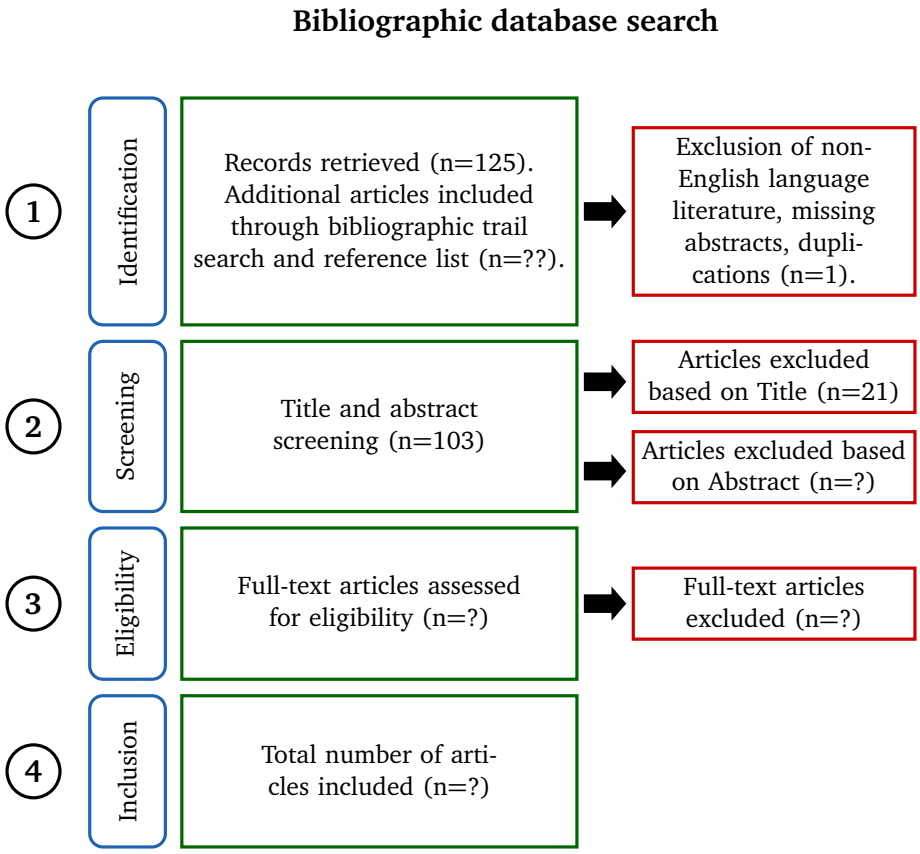

Figure 4.1: Flowchart of search and selection SLR strategy.

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

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

Chapter 5

Research results

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

5.1 Descriptive analysis

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

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

Many tools and methods for vulnerability detection have been developed over the recent years. These tools can be categorized in terms of their primary function. Primarily, they can be divided into three categories: Static Vulnerability Detection, Dynamic Vulnerability Detection, and Machine Learning based Vulnerability Detection. In the following sections the identified vulnerability detection tools are summarized, compared and analyzed in detail (up to December 2021).

The following table lists the tools and their respective categories.

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

Add snowballing

5.2.1 Static Vulnerability Detection Methods

Static vulnerability detection methods are performed without actually executing programs. The analysis is performed by examining the source code or bytecode of the smart contract.

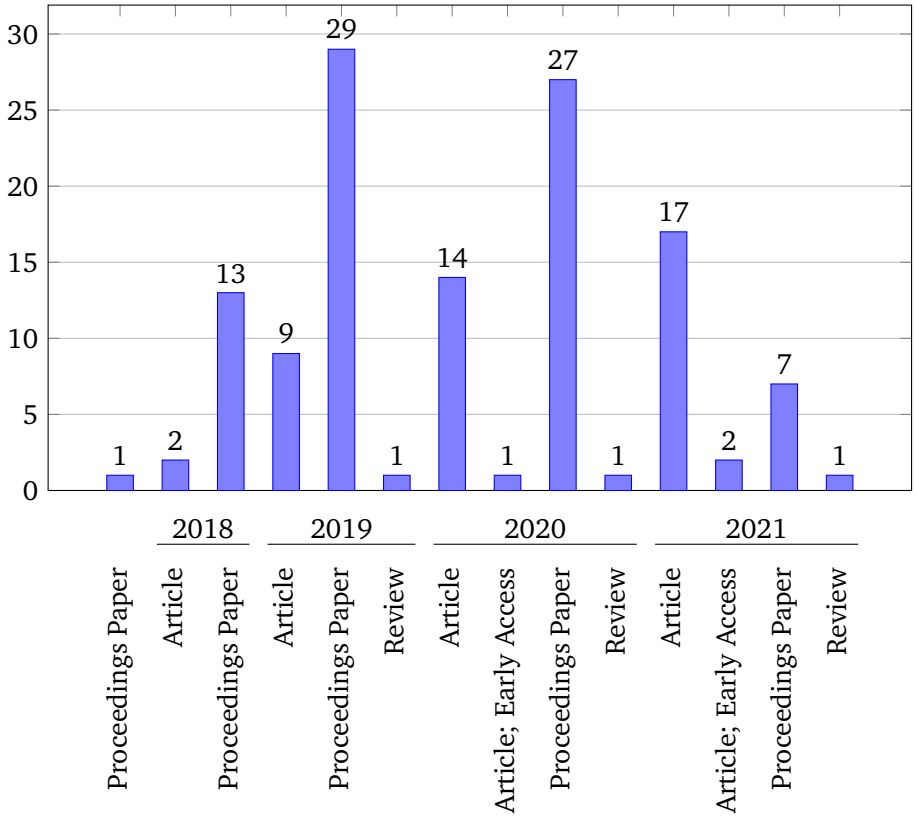

Figure 5.1: Distribution of selected literature type by year.

Table 5.1: Vulnerabilities enumeration.

#	**Vulnerability name**	#	**Vulnerability name**
1	Integer Overflow	4	Stack Size Limit (deprecated)
2	Integer Underflow	5	Timestamp Dependency
3	Transaction-Ordering Dependence	6	Reentrancy

Table 5.2: Comparison of vulnerability types detectable by existing tools.

Tool	**Vulnerability**					
	1	2	3	4	5	6
ContractWard	✓	✓	✓	✓	✓	✓
ESCORT						
Zeus	✓	✓	✓		✓	✓
OYENTE	✓	✓		✓	✓	✓
Maian[1]	✗	✗	✗	✗	✗	✗
Manticore[2]	-	-	-	-	-	-
Mythril	✓	✓	✓	✓	✓	✓
Dedaub	?	?	?	?	?	✓
Securify	✗	✗	✓	✗	?	✓
Vandal	✗	✗	✗	✗	✗	✓
Towards Sequential[3]	✗	✗	✗	✗	✗	✗
NLP-inspried[4]	✗	✗	✗	✗	✗	✗
Color-inspriedtnote5	✓	✓	✓	✓	✓	✓
Graph NN-based	✗	✗	✗	✓[6]	✓	✓

[1] Detects Greedy, suicidal and prodigal contracts.
[2] Focuses on maximizing code coverage. Can be somewhat extended to detect more bugs/vulnerabilities.
[3] Based on Maian.
[4] Categories are Suicidal, Prodigal, Greedy, Normal Smart Contracts, and "Prodigal and Greedy".
[5] Authors don't specifically list the detectable vulnerabilities.
[6] Expressed as infinite loop vulnerabilities. This is also problematic for gas consumption.

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

change to 1.2textwidth ?

Symbolic Execution

Symbolic execution is a ...

OYENTE. OYENTE is one of the earliest Ethereum SC vulnerability detection tool, developed by Luu *et al.* [25] in 2016. The tool consists of four main components, named CFGBuilder, Explorer, CoreAnalysis, and Validator. The CFGBuilder component builds a Control Flow Graph (CFG) of a smart contract. The Explorer component is a symbolic execution engine that explores the CFG, using the Z3 constraint solver for determining if a branch condition is either provably true or provably false along the current path. The CoreAnalysis component analyzes the explored CFG to detect vulnerabilities. The Validator component is finally used for eliminating false positives. OYENTE uses EVM bytecode as its input. OYENTE has paved the way for a lot of subsequent research. It is able to detect Call Stack Risk, Reentrancy Risk, Transaction Order Risk and Timestamp Risk.

ETHIR. Albert *et al.* [6] analyzes EVM bytecode based on the rule-based representations of CFG produced by an improved version of OYENTE [25]. The improvement primarily consist of including all possible jump addresses, whereas originally OYENTE only stores the last jump address. This allows ETHIR to produce a complete CFG, containing both control-flow and data-flow information of the SC EVM bytecode. From this, ETHIR generates guarded rules for checking for conditional and unconditional jump instructions. This enables the application of (existing) high-level analyses to infer properties of the EVM code. Albert *et al.* [6] uses the high-level static analyzer SACO (Static Analyzer for Concurrent Objects) for checking conditional and unconditional jump instructions.

SAFEVM. SAFEVM is an verification tool for Ethereum smart contracts that makes use of existing verification engines for C programs. Albert *et al.* [7] uses OYENTE and ETHIR in order to produce a rule-based representation of EVM bytecode or Solidity. This is then converted into a special C program. Existing analysis tools are then used to verify the security of the converted C program, and a report with the verification results is outputted.

SASC. SASC [8] 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.

Table 5.3: Existing static smart contract vulnerability detection tools.

Refs.	Year	Name	Assistive technology	Capability	Input[a]
Symbolic execution					
[5]	2016	OYENTE	–	Multi-class	Solidity*, EVM bytecode
[6]	2018	ETHIR	–	General-purpose (depends on high-level verifier)	Solidity*, EVM bytecode
[7]	2019	SAFEVM	Symbolic model checking	General-purpose (depends on C verifier)	Solidity*, EVM bytecode
[8]	2018	SASC	Topological analysis and syntax analysis	Multi-class	Solidity
[9]	2018	Mythril	Taint analysis and symbolic model checking	Multi-class	EVM bytecode
[10]	2018	MPro	Taint analysis, symbolic model checking and data-flow analysis	Multi-class	EVM bytecode
[11]	2018	Maian	Concrete validation	Multi-class	Solidity, EVM bytecode
[12]	2018	SECURIFY	Abstract interpretation	Binary decision	Solidity*, EVM bytecode
[13]	2019	SolAnalyser	Code instrumentation and execution trace analysis	Multi-class	Solidity
[14]	2019	Manticore	–		EVM bytecode
[15]	2020	RA	–	Reentrancy	EVM bytecode
Syntax analysis					
[16]	2018	SmartCheck	Topological analysis	Multi-class	Solidity
[17]	2019	NeuCheck	–	Multi-class	Solidity
[18]	2020	ReJection	–	Reentrancy	Solidity
Abstract interpretation					
[19]	2018	Zeus	Symbolic model checking		Source code
[20]	2018	Vandal	Logic-driven analysis	Multi-class	Bytecode
[21]	2018	MadMax	Decompilation	Multi-class gas-related	EVM bytecode
Data Flow Analysis					
[22]	2019	Slither	Taint analysis	Multi-class	Solidity
[23]	2020	CLAIRVOYANCE	Taint analysis	Reentrancy	Solidity
[24]	2020	SESCon	Taint analysis and Syntax analysis	Multi-class	Solidity

[a] "*" means the source code is compiled down to bytecode.

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

Is this symbolic execution, or trace analysis?

Mythril. Mythril, developed by Mueller [9] is a command line tool, combining symbolic execution, with taint analysis and SMT. Mythril takes in EVM bytecode as input. In addition, Mythril also provides interfaces to allow for developers to write custom vulnerability detection logic. In order to cover the situation where a contract is called upon multiple times, Mythril simulates this through conducting multiple symbolic executions (two by default).

MPro. MPro by Zhang *et al.* [10] is a fully automated and scalable security scanning tool for Ethereum SC. MPro combines the existing tools Mythril [9] and Slither [22], leveraging both static and symbolic analysis to prune unnecessary execution paths and achieve n times efficiency increase than that of Mythril.

MAIAN. MAIAN [11] is a symbolic execution tool for specifying and inferring trace vulnerabilities. Nikolić *et al.* [11] points out that most existing tools ignore problems related to calling a contract multiple times. Contracts containing trace vulnerabilities could: lock funds indefinitely; allow for transfer founds to any address; be killed by anyone. Based on these attributes, MAIAN marks vulnerable contracts in three categories, greedy, prodigal, and suicidal. MAIAN takes EVM bytecode as input. Along with user-defined analysis targets, this allows for confirming the presence of a trace vulnerability.

SECURIFY. SECURIFY[12] is a security analysis tool for Ethereum SC that combines abstract interpretation with symbolic execution. Thee tool automatically classifies behaviors of a contract into three categories, compliance (matched by compliance properties), violation (matched by violation properties), and warning (matched by neither). SECURIFY takes in EVM bytecode as input, along with a security model defined with a new domain-specific language. The use of a domain-specific language, enables users to express new vulnerability patterns as they emerge.

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

Investigate the effort neeeded to implement an other execution environment.

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

Syntactical analysis

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

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

Findsomee way to formulate this...

NeuCheck. Lu *et al.* [17] introduces NeuCheck, a cross-platform SC syntax analysis tool for Ethereum. NeuCheck generates a syntax tree from Solidity source code, and generates an XML-based IR. The open-source tool dom4j [1] is then leveraged for parsing this tree and completing the analysis.

ReJection. Ma *et al.* [18] purpose a method for detecting SC reentrancy vulnerability detection based on analysis of ASTs. The analysis conducted by ReJection is comprised of four steps. An AST is generated from Solidity source code with the SC compiler solc [2]. 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.

Abstract interpretation

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

Rewrite this info.

[1] `https://dom4j.github.io`
[2] `https://github.com/ethereum/solidity`

Zeus. Zeus purposed by Kalra *et al.* [19] combines both abstract interpretation, symbolic model checking and constrained horn clauses. Model checking is verification method where a a system is modeled into a finite state machine. This is then used for verifying whether the system meets certain criteria. Zeus takes Solidity SC code as input, and translates it into an low-level IR called LLVM-IR. LLVM is a compiler toolchain, that supports a large ecosystem of code analysis tools. Along with the LLVM code, Zeus also requires a user to provide a set of policies that are used in order to .... Finally, existing LLVM verification based tools are used on the constrained horn clauses to identify vulnerabilities.

refs..

include gray litterature - Zeus!!!

MadMax. MadMax is a gas-focused vulnerability detection tool by Grech *et al.* [21]. The authors creates a pipeline consisting of a control flow analysis based decompiler that disassembles EVM bytecode into a structured low-level IR. This is then analyzed in DataLog [26] by representing the IR as DataLog [26] 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.

Data Flow Analysis

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

Rewrite this info.

Slither. Slither [22] 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 [23]. The tool models cross-function and cross-contract behaviors, in order to enable a more sound analysis than Slither [22], OYENTE [25] and SECURIFY [12]. CLAIRVOYANCE applies cross-contract static taint analysis to find reentrancy candidates, then integrates path protective techniques to refine the results. The authors claim that the tool significantly outperforms the three static tools in terms of precision and accuracy.

SESCon. Ali *et al.* [24] 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.* [24] claims that SESCon outperforms

other analyzers and can accurately detect up to 90% of known vulnerability patterns.

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

5.2.2 Formal Verification

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

Rewisit this info.

5.2.3 Dynamic Vulnerability Detection Methods

Dynamic program analysis is a form of analysis performed by executing programs on a real or virtual processor. This is in contrast to static program analysis. In order to detect vulnerabilities, the program is monitored during execution. However, in order to be effective, sufficient inputs needs to be provided/tested.

Table 5.4: Existing dynamic smart contract vulnerability detection tools.

Method	Name	Assistive technology	Capability	Input
????	Dedaub	Flow and loop analysis	Gas-focused vulnerability	Sources code
	??	??	??	??
	??	??	??	??
	??	??	??	??
	??	??	??	??
	??	??	??	??
Fuzzing	??	??	??	???
	??	??	??	??
	??	??	??	??
	??	??	??	??
	??	??	??	??
	??	??	??	??

Fuzzing

MythX ReGuard (Reentrancy) ContractFuzzer (ABI specifications) Echidna ILF

Validation

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

5.2.4 Machine Learning for Vulnerability Detection

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

Table 5.5: Existing ML-based smart contract vulnerability detection tools.

Name	Method	Capability	Input
ContractWard	Bigram model	Binary decision	Opcode
ESCORT	LSTM + transfer learning	Multi-label	Bytecode
Towards Sequential	LSTM	Binary decision	Opcode
NLP-inspried [27]	AWD-LSTM	Multi-class	Opcode
Color-inspried [28]	CNN	Multi-label	Bytecode
Graph NN-based [29]	GNN	Multi-class	Source code

Long Short-Term Memory

Average Stochastic Gradient Descent Weighted Dropped LSTM

Convolutional Neural Network

Graph Neural Network

AWD-LSTM based

Determine categoory of ML-based tools.

Use multi class or binary decision categories?

Use the input as sections?, Eg. AST, byte-

ContractWard

ContractWard [30] implements a smart contract vulnerability detection tool based on machine learning. It is a state-of-the-art tool for detecting smart contract vulnerabilities. It is a multi-label classifier that can detect multiple vulnerabilities. The method is based on a bigram model. The input is the bytecode of the smart contract. The output is a binary decision. The decision is 1 if the contract is vulnerable, 0 otherwise.

Models for Ethereum Smart Contracts Uses XGBoost as multi label classifier with SMOTETomke as sampling method. ses simplified smar contract opcodes as

input. the static opcodes represent static featturesof contracts. the tool is appropriate for rapid batch detection of vulnerabilities in smart contracts, with an average detection speed of 4 seconds per contract. Reliable with Micro-F1 and Macro-F1 over 96%. Only operates on opcodes as input. simplified smart contract opcodes are extracted from the bytecode of smart contracts.

ESCORT

ESCORT provides a long short-term memory machine learning model in order to detect smart contract vulnerabilities. ContractScraper is used to extract the bytecode of the smart contract. The model is based on a long short-term memory (LSTM) network. The model is trained on the Ethereum blockchain. The model is able to detect multiple vulnerabilities. The model is capable of detecting the following vulnerabilities:

"ESCORT, the first Deep Neural Network (DNN)-based vulnerability detection framework for Ethereum smart contracts that supports lightweight transfer learning on unseen security vulnerabilities, thus is exten- sible and generalizable"

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

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

Kalra *et al.* [19] provides a tool called Zeus. Zeus translates the Solidity SC language into LLVM-IR language. LLVM is a compiler toolchain, that supports a large ecosystem of code analysis tools. However, the LLVM-IR is very is an IR that that can be used to detect vulnerabilities in the SCs language.

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

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

The most promising tool is Zeus by Kalra *et al.* [19] that allows for cross-chain compatibility through the use of LLVM. By using LLVM, any SC language can be translated into the LLVM language. This is also the only existing tool that also demonstrates cross-chain support through a prototypal mock implementation for Hyperledger Fabric [4].

Another solution is the SC language SCILLA by Sergey *et al.* [31]. This is supposed to be an type of intermediate language based on communicating automata. However, there are no automatic translation tools for converting another language

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

into SCILLA. Other similar solutions include converting Solidity into F*; a general-purpose functional programming language with effects aimed at program verification [32].

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

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

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

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

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

Chapter 6

Discussion

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

Several of the tools and

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

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

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

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

It also seems that most research

Possible to detect ponzi schemes?

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

6.1 Threats to Validity

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

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

Chapter 7

Conclusion

include recommendations.

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

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

Bibliography

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

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

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

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

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

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

[7] E. 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}.

[8] 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.

[9] B. Mueller. "Mythx tech: Behind the scenes of smart contract security analysis." (Apr. 2018), [Online]. Available: https://blog.mythx.io/features/mythx-%20tech-%20behind-%20the-%20scenes-%20of-%20smart-%20contract-%20analysis/ (visited on 10/27/2021).

[10] 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}.

[11] 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.

[12] 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.

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

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

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

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

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

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

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

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

[20] 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].

[21] 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.

[22] 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.

[23] 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.

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

[25] 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}.

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

[27] 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].

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

[29] 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.

[30] 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.

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

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

[33] K. K. Landes, “A scrutiny of the abstract,” *Bulletin of the American Association of Petroleum Geologists*, vol. 35, no. 7, p. 1660, 1951.

Paper I

Here, you may add a description of the paper, an illustration, or just give the bibliographic reference:

> K. K. Landes, “A scrutiny of the abstract,” *Bulletin of the American Association of Petroleum Geologists*, vol. 35, no. 7, p. 1660, 1951

Or you may leave it empty, if you like.

BULLETIN OF THE AMERICAN ASSOCIATION OF PETROLEUM GEOLOGISTS
VOL. 50, NO. 9 (SEPTEMBER, 1966), P. 1992-1999, 2 FIGS.

GEOLOGICAL NOTES

A SCRUTINY OF THE ABSTRACT, II[1]

KENNETH K. LANDES[2]
Ann Arbor, Michigan

ABSTRACT

A partial biography of the writer is given. The inadequate abstract is discussed. What should be covered by an abstract is considered. The importance of the abstract is described. Dictionary definitions of "abstract" are quoted. At the conclusion a revised abstract is presented.

For many years I have been annoyed by the inadequate abstract. This became acute while I was serving a term as editor of the *Bulletin* of The American Association of Petroleum Geologists. In addition to returning manuscripts to authors for rewriting of abstracts, I also took 30 minutes in which to lower my ire by writing, "A Scrutiny of the Abstract."[1] This little squib has had a fantastic distribution. If only one of my scientific outpourings would do as well! Now the editorial board of the Association has requested a revision. This is it.

The inadequate abstract is illustrated at the top of the page. The passive voice is positively screaming at the reader! It is an outline, with each item in the outline expanded into a sentence. The reader is told what the paper is about, but not what it contributes. Such abstracts are merely overgrown titles. They are produced by writers who are either (1) beginners, (2) lazy, or (3) have not written the paper yet.

To many writers the preparation of an abstract is an unwanted chore required at the last minute by an editor or insisted upon even before the paper has been written by a deadline-bedeviled program chairman. However, in terms of market reached, the abstract is *the most important part of the paper*. For every individual who reads or listens to your entire paper, from 10 to 500 will read the abstract.

If you are presenting a paper before a learned society, the abstract alone may appear in a preconvention issue of the society journal as well as in the convention program; it may also be run by trade journals. The abstract which accompanies a published paper will most certainly reappear in abstract journals in various languages, and perhaps in company internal circulars as well. It is much better to please than to antagonize this great audience. Papers written for oral presentation should be *completed prior to the deadline for the abstract*, so that the abstract can be prepared from the written paper and not from raw ideas gestating in the writer's mind.

My dictionary describes an abstract as "a summary of a statement, document, speech, etc. . . ." and that which *concentrates in itself the essential information* of a paper or article. The definition I prefer has been set in italics. May all writers learn the art (it is not easy) of preparing an abstract containing the *essential information* in their compositions. With this goal in mind, I append an abstract that should be an improvement over the one appearing at the beginning of this discussion.

ABSTRACT

The abstract is of utmost importance, for it is read by 10 to 500 times more people than hear or read the entire article. It should not be a mere recital of the subjects covered. Expressions such as "is discussed" and "is described" should *never* be included! The abstract should be a condensation and concentration of the *essential information* in the paper.

[1] Revised from K. K. Landes' "A Scrutiny of the Abstract," first published in the *Bulletin* in 1951 (*Bulletin*, v. 35, no. 7, p. 1660). Manuscript received, June 3, 1966; accepted, June 10, 1966.

Editor's note: this abstract is published together with The Royal Society's "Guide for Preparation and Publication of Abstracts" to give *Bulletin* authors two viewpoints on the writing of abstracts.

[2] Professor of geology and mineralogy, University of Michigan. Past editor of the *Bulletin*.

Appendix A

Additional Material

Additional material that does not fit in the main thesis but may still be relevant to share, e.g., raw data from experiments and surveys, code listings, additional plots, pre-project reports, project agreements, contracts, logs etc., can be put in appendices. Simply issue the command `\appendix` in the main `.tex` file, and make one chapter per appendix.

If the appendix is in the form of a ready-made PDF file, it should be supported by a small descriptive text, and included using the `pdfpages` package. To illustrate how it works, a standard project agreement (for the IE faculty at NTNU in Gjøvik) is attached here. You would probably want the included PDF file to begin on an odd (right hand) page, which is achieved by using the `\cleardoublepage` command immediately before the `\includepdf[]{}` command. Use the option `[pages=-]` to include all pages of the PDF document, or, e.g., `[pages=2-4]` to include only the given page range.

NTNU

Vår dato Vår referanse

Norges teknisk-naturvitenskapelige universitet

Prosjektavtale

mellom NTNU Fakultet for informasjonsteknologi og elektroteknikk (IE) på Gjøvik (utdanningsinstitusjon), og

_____

_____ (oppdragsgiver), og

_____

_____

_____ (student(er))

Avtalen angir avtalepartenes plikter vedrørende gjennomføring av prosjektet og rettigheter til anvendelse av de resultater som prosjektet frembringer:

1. Studenten(e) skal gjennomføre prosjektet i perioden fra _____ til_____ .

   Studentene skal i denne perioden følge en oppsatt fremdriftsplan der NTNU IE på Gjøvik yter veiledning. Oppdragsgiver yter avtalt prosjektbistand til fastsatte tider. Oppdragsgiver stiller til rådighet kunnskap og materiale som er nødvendig for å få gjennomført prosjektet. Det forutsettes at de gitte problemstillinger det arbeides med er aktuelle og på et nivå tilpasset studentenes faglige kunnskaper. Oppdragsgiver plikter på forespørsel fra NTNU å gi en vurdering av prosjektet vederlagsfritt.

2. Kostnadene ved gjennomføringen av prosjektet dekkes på følgende måte:
   - Oppdragsgiver dekker selv gjennomføring av prosjektet når det gjelder f.eks. materiell, telefon/fax, reiser og nødvendig overnatting på steder langt fra NTNU på Gjøvik. Studentene dekker utgifter for ferdigstillelse av prosjektmateriell.
   - Eiendomsretten til eventuell prototyp tilfaller den som har betalt komponenter og materiell mv. som er brukt til prototypen. Dersom det er nødvendig med større og/eller spesielle investeringer for å få gjennomført prosjektet, må det gjøres en egen avtale mellom partene om eventuell kostnadsfordeling og eiendomsrett.

3. NTNU IE på Gjøvik står ikke som garantist for at det oppdragsgiver har bestilt fungerer etter hensikten, ei heller at prosjektet blir fullført. Prosjektet må anses som en eksamensrelatert oppgave som blir bedømt av intern og ekstern sensor. Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.

4. Alle bacheloroppgaver som ikke er klausulert og hvor forfatteren(e) har gitt sitt samtykke til publisering, kan gjøres tilgjengelig via NTNUs institusjonelle arkiv hvis de har skriftlig karakter A, B eller C.

Tilgjengeliggjøring i det åpne arkivet forutsetter avtale om delvis overdragelse av opphavsrett, se «avtale om publisering» (jfr Lov om opphavsrett). Oppdragsgiver og veileder godtar slik offentliggjøring når de signerer denne prosjektavtalen, og må evt. gi skriftlig melding til studenter og instituttleder/fagenhetsleder om de i løpet av prosjektet endrer syn på slik offentliggjøring.

Den totale besvarelsen med tegninger, modeller og apparatur så vel som programlisting, kildekode mv. som inngår som del av eller vedlegg til besvarelsen, kan vederlagsfritt benyttes til undervisnings- og forskningsformål. Besvarelsen, eller vedlegg til den, må ikke nyttes av NTNU til andre formål, og ikke overlates til utenforstående uten etter avtale med de øvrige parter i denne avtalen. Dette gjelder også firmaer hvor ansatte ved NTNU og/eller studenter har interesser.

5. Besvarelsens spesifikasjoner og resultat kan anvendes i oppdragsgivers egen virksomhet. Gjør studenten(e) i sin besvarelse, eller under arbeidet med den, en patentbar oppfinnelse, gjelder i forholdet mellom oppdragsgiver og student(er) bestemmelsene i Lov om retten til oppfinnelser av 17. april 1970, §§ 4-10.

6. Ut over den offentliggjøring som er nevnt i punkt 4 har studenten(e) ikke rett til å publisere sin besvarelse, det være seg helt eller delvis eller som del i annet arbeide, uten samtykke fra oppdragsgiver. Tilsvarende samtykke må foreligge i forholdet mellom student(er) og faglærer/veileder for det materialet som faglærer/veileder stiller til disposisjon.

7. Studenten(e) leverer oppgavebesvarelsen med vedlegg (pdf) i NTNUs elektroniske eksamenssystem. I tillegg leveres ett eksemplar til oppdragsgiver.

8. Denne avtalen utferdiges med ett eksemplar til hver av partene. På vegne av NTNU, IE er det instituttleder/faggruppeleder som godkjenner avtalen.

9. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og NTNU som regulerer nærmere forhold vedrørende bl.a. eiendomsrett, videre bruk, konfidensialitet, kostnadsdekning og økonomisk utnyttelse av resultatene. Dersom oppdragsgiver og student(er) ønsker en videre eller ny avtale med oppdragsgiver, skjer dette uten NTNU som partner.

10. Når NTNU også opptrer som oppdragsgiver, trer NTNU inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.

11. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene imellom. Dersom det ikke oppnås enighet, er partene enige om at tvisten løses av voldgift, etter bestemmelsene i tvistemålsloven av 13.8.1915 nr. 6, kapittel 32.

12. Deltakende personer ved prosjektgjennomføringen:

NTNUs veileder (navn): _____

Oppdragsgivers kontaktperson (navn): _____

Student(er) (signatur): _____ dato _____

_____ dato _____

_____ dato _____

_____ dato _____

Oppdragsgiver (signatur):_____ dato _____

Signert avtale leveres digitalt i Blackboard, rom for bacheloroppgaven.
Godkjennes digitalt av instituttleder/faggruppeleder.

Om papirversjon med signatur er ønskelig, må papirversjon leveres til instituttet i tillegg.
Plass for evt sign:

Instituttleder/faggruppeleder (signatur): _____ dato _____