Research Project: Smart Contract Vulnerability Detection Methods

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

Abstract . . . . **i**
Sammendrag . . . . **ii**
Contents . . . . **iii**
Figures . . . . **v**
Tables . . . . **vi**
Code Listings . . . . **vii**
Acronyms . . . . **viii**
Glossary . . . . **ix**
Todo list . . . . **x**
1 Introduction **1**
2 Background **2**
2.1 Blockchain 2
2.2 Smart Contracts 2
2.3 Ethereum 2
2.4 Smart Contract Security Vulnerabilities 3
2.4.1 Integer Overflow and Underflow 3
2.4.2 Transaction-Ordering Dependence 3
2.4.3 Stack Size Limit 3
2.4.4 Timestamp Dependency 3
2.4.5 Reentrancy 3
2.4.6 Unfair Contracts 4
3 Related work **5**
3.1 General security frameworks 5
4 Research Method **6**
4.1 Research Motivation 6
4.2 Research Questions 6
4.3 Research Method and Design 6
4.3.1 Locating Studies 6
4.3.2 Study Selection and evaluation 7
4.3.3 Analysis and Synthesis 7
5 Research results **9**
5.1 Research Question 1: What are the current approaches for Smart Contract vulnerability detection? 9
5.1.1 Static Vulnerability Detection Methods 9

5.1.2 Dynamic Vulnerability Detection Methods 11
5.1.3 Machine Learning for Vulnerability Detection 11
5.2 Research Question 2: What is the current research on cross-chain Smart Contract vulnerability detection? 13
5.3 Research Question 3: How to make Smart Contract vulnerability detecting possible cross-chain? 13
6 Discussion 15
7 Conclusion 16
Bibliography 17
Paper I 19
A Additional Material 21

Figures

4.1 Flowchart of search and selection SLR strategy. 8

Tables

4.1 Existing ML-based smart contract vulnerability detection tools. . . . 7

5.1 Vulnerabilities enumeration. 9
5.2 Comparison of vulnerability types detectable by existing tools. . . . 10
5.3 Existing static smart contract vulnerability detection tools. 10
5.4 Existing dynamic smart contract vulnerability detection tools. 11
5.5 Existing ML-based smart contract vulnerability detection tools. . . . 12

Code Listings

2.1 Timestamp Dependency vulnerabile Solidity Smart Contract code . 3
2.2 Timestamp Dependency vulnerabile Solidity Smart Contract code . 4

Acronyms

IR Intermediate Representation. viii, 13, *Glossary:* Intermediate Representation

ML Machine Learning. 13, 14

NFT Non Fungible Tokens. viii, 2, *Glossary:* Non Fungible Tokens

SC Smart Contract. iii, iv, vii, ix, 2–6, 9, 13–15

SLR Systematic Literature Review. v, 6, 8

WoS Web of Science. viii, 6, 7, *Glossary:* Web of Science

Glossary

BibTeX File format for storing bibliographic information. 7

F* General-purpose functional programming language with effects aimed at program verification. 13

gas Gas in crypto refers to the computational effort required to execute operations. This is normally paid in the blockchain's plattform cryptocurrency. 10

Intermediate Representation A representation for use as an intermediate step. 13

LLVM A compiler toolchain providing a complete infrastructure for creating compiler frontends and backends. 13

LLVM-IR LLVM-IR is the intermediate representation of the LLVM compiler toolchain. 13

Non Fungible Tokens A type of token that is unique. 2

Solidity Solidity is a Smart Contract language developed for Ethereum. 13

Web of Science Scientific citation database. 6, 7

Todo list

call it paper or document? 1
Include articles retrievedby sources. 7
Majority of the tables, and existing vulnerability tools needs to be describeed here. 9
Determine categoory of ML-based tools. 12
Use multi class or binary decision categories? 12
Use the input as sections?, Eg. AST, bytecode, source code, llvm, etc. 12
Zeus: Emphasize on the need for policies and that it is a prototype implementation. 13

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 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.

2.2 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 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.

2.4 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:

2.4.1 Integer Overflow and Underflow

2.4.2 Transaction-Ordering Dependence

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

2.4.3 Stack Size Limit

2.4.4 Timestamp Dependency

Considering changing to "Block state dependence"

Code listing 2.1: Timestamp Dependency vulnerabile Solidity Smart Contract code

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

2.4.5 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: Timestamp Dependency vulnerabile Solidity Smart Contract code

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

2.4.6 Unfair Contracts

Chapter 3

Related work

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

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.

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 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.

4.2 Research Questions

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 in this Systematic Literature Review (SLR) are based on the following principles by ....:

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" vulnerability AND (detection OR analysis))

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

	Stage	**Selection criteria**
Inclusion	Identification	Non english articles, missing abstracts, notes or editorials.
	Screening	col3
Exclusion	Eligibility	col3
	Inclusion	Inclusion
	col2	col3
	col2	col3

No time restrictions was set. The search results were exported to a BibTeX file and processed locally.

Include articles retrievedby sources.

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.

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, notes or editorials. During screening, ? 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.

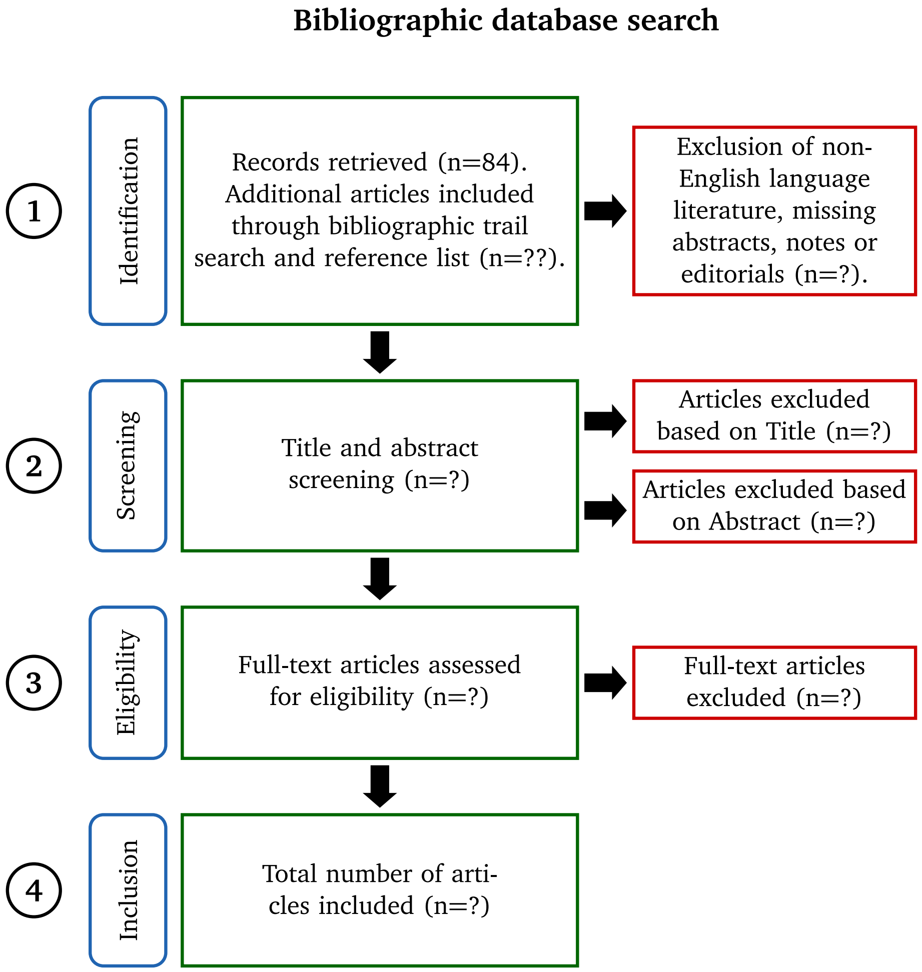
Bibliographic database search
1
Identification
Records retrieved (n=84). Additional articles included through bibliographic trail search and reference list (n=??).
Exclusion of non-English language literature, missing abstracts, notes or editorials (n=?).
2
Screening
Title and abstract screening (n=?)
Articles excluded based on Title (n=?)
Articles excluded based on Abstract (n=?)
3
Eligibility
Full-text articles assessed for eligibility (n=?)
Full-text articles excluded (n=?)
4
Inclusion
Total number of articles included (n=?)

Figure 4.1: Flowchart of search and selection SLR strategy.

Chapter 5

Research results

The results chapter should simply present the results of applying the methods presented in the method chapter without further ado. This chapter will typically contain many graphs, tables, etc. Sometimes it is natural to discuss the results as they are presented, combining them into a ‘Results and Discussion’ chapter, but more often they are kept separate.

There are many existing methods and tools for vulnerability detection. These can be divided up into three categories: Static Vulnerability Detection, Dynamic Vulnerability Detection, and Machine Learning based Vulnerability Detection. The following table lists the tools and their respective categories.

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

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

Majority of the tables, and existing vulnerability tools needs to be describeed here.

5.1.1 Static Vulnerability Detection Methods

"Static detection techniques analyze the smart contract in a static environment by examining its source code or bytecode."

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.

Table 5.3: Existing static smart contract vulnerability detection tools.

Name	Method	Capability	Required input
Zeus [3]			
Oyente	Symbolic execution	Multi-class	Source code and bytecode
Maian	Symbolic analysis	Multi-class	Source code and bytecode
Manticore [4]	Symbolic execution	Code coverage	Bytecode
Mythril	Symbolic execution, taint analysis, and SMT	Multi-class	Bytecode
Securify [5]	Symbolic analysis	Binary decision	Bytecode
Vandal [6]	Logic-driven static program analysis	Multi-class	Bytecode

Information Flow Analysis-based Vulnerability Detection

Symbolic Execution-based

Oyente is a symbolic execution tool for smart contract analysis. manticore, Securify, teEther

Information Flow Analysis-based

Slither (taint analysis), Dedaubb

5.1.2 Dynamic Vulnerability Detection Methods

"Dynamic testing techniques execute the program and observe its behaviors to determine the vulnerability's existence."

Table 5.4: Existing dynamic smart contract vulnerability detection tools.

Name	Method	Capability	Required input
Dedaub	Flow and loop analysis	Gas-focused vulnerability	Sources code

Fuzzing-based

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

Validation-based

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

5.1.3 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	Required input
ContractWard	ML (bigram model)	Binary decision	Opcode
ESCORT	ML (LSTM) + transfer learning	Multi-label	Bytecode
Towards Sequential	ML (LSTM)	Binary decision	Opcode
NLP-inspried [7]	ML (AWD-LSTM)	Multi-class	Opcode
Color-inspried [8]	ML (CNN)	Multi-label	Bytecode
Graph NN-based [9]	ML (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, bytecode, source code, llvm, etc.

ContractWard

ContractWard [10] 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"

5.2 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.* [3] 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.* [3] 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 [11].

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

Another solution is the SC language SCILLA by Sergey *et al.* [12]. This is supposed to be an type of intermediate language based on communicating automata. However, there are no automatic translation tools for converting another language into SCILLA. Other similar solutions include converting Solidity into F*; a general-purpose functional programming language with effects aimed at program verification [13].

5.3 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

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 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).

Chapter 7

Conclusion

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

[4] 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 *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2019, pp. 1186–1189. DOI: `10.1109/ASE.2019.00133`.

[5] 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`.

[6] 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]`.

[7] 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]`.

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

[9] 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.

[10] 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.

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

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

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

[14] 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.

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 ____________