SoK: A Unified Data Model for Smart Contract Vulnerability Taxonomies

Claudia Ruggiero
claudia.ruggiero@uniroma1.it
Sapienza Università di Roma
Rome, Italy

Pietro Mazzini
pietro.mazzini@uniroma1.it
Sapienza Università di Roma
Rome, Italy

Emilio Coppa
ecoppa@luiss.it
LUISS University
Rome, Italy

Simone Lenti
simone.lenti@uniroma1.it
Sapienza Università di Roma
Rome, Italy

Silvia Bonomi
silvia.bonomi@uniroma1.it
Sapienza Università di Roma
Rome, Italy

ABSTRACT

Modern blockchains support the execution of application-level code in the form of *smart contracts*, allowing developers to devise complex Distributed Applications (DApps). Smart contracts are typically written in high-level languages, such as Solidity, and after deployment on the blockchain, their code is executed in a distributed way in response to transactions or calls from other smart contracts. As a common piece of software, smart contracts are susceptible to vulnerabilities, posing security threats to DApps and their users.

The community has already made many different proposals involving taxonomies related to smart contract vulnerabilities. In this paper, we try to systematize such proposals, evaluating their common traits and main discrepancies. A major limitation emerging from our analysis is the lack of a proper formalization of such taxonomies, making hard their adoption within, e.g., tools and disfavoring their improvement over time as a community-driven effort. We thus introduce a novel data model that clearly defines the key entities and relationships relevant to smart contract vulnerabilities. We then show how our data model and its preliminary instantiation can effectively support several valuable use cases, such as interactive exploration of the taxonomy, integration with security frameworks for effective tool orchestration, and statistical analysis for performing longitudinal studies.

CCS CONCEPTS

• **General and reference** → **Surveys and overviews**; • **Security and privacy** → **Distributed systems security**.

KEYWORDS

Smart Contract, Vulnerability, Blockchain, Weakness, Taxonomy

ACM Reference Format:
Claudia Ruggiero, Pietro Mazzini, Emilio Coppa, Simone Lenti, and Silvia Bonomi. 2024. SoK: A Unified Data Model for Smart Contract Vulnerability Taxonomies. In *The 19th International Conference on Availability, Reliability and Security (ARES 2024), July 30–August 02, 2024, Vienna, Austria.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3664476.3664507

ARES 2024, July 30–August 02, 2024, Vienna, Austria

ACM ISBN 979-8-4007-1718-5/24/07
https://doi.org/10.1145/3664476.3664507

1 INTRODUCTION

Distributed Ledger Technologies (DLTs) attracted a lot of attention in the last decade due to the spreading of blockchains [9, 53] i.e., a particular class of DLTs where transactions are stored in blocks replicated over a network of participating nodes and cryptographically linked together to capture and preserve causality relationships. Each block contains a set of transactions, a timestamp, and a reference to (i.e., a hash of the content of) the previous block, offering a consistent, secure and tamper-evident record of all transactions.

Modern blockchain technologies extended their functionalities to the application layer by supporting the execution of *Decentralized Applications* (DApps) currently adopted in many different domains [94], ranging from finance to agriculture. DApps are implemented through *smart contracts*, i.e., pieces of software written in a high-level programming language (e.g., in Solidity [27]) encoding the *rules* behind the application logic and deployed on top of the blockchain. When deployed on the blockchain, the smart contract code is executed by a *Virtual Machine* (VM), replicated on the blockchain nodes, enforcing the terms of the contract when predefined conditions are met. This decentralization ensures that the contract's execution is transparent, secure, and resistant to manipulation. While today there are several blockchain technologies offering support for smart contracts execution, the majority of them are currently compatible with the Ethereum Virtual Machine (EVM)[1] and support smart contracts written in Solidity letting them become a *de facto standard*. Being a public and open technology, Ethereum requires its users to commit resources for the maintenance of the overall system. In particular, to make Ethereum sustainable, every operation on the network (and thus every action in the smart contract) requires a certain amount of resources that are measured in *gas units*. Users must pay such gas to incentivize the execution of their smart contracts, limiting the probability of denial-of-service attacks.

Being the means to develop distributed and decentralized applications, smart contracts have a vast potential but, as software entities, they are not free from bugs and flaws exposing them to vulnerabilities potentially leading to highly impacting attacks. As an

[1]The EVM is the virtual machine originally introduced by Ethereum [9] making it practically the first *programmable* blockchain.

example, let us recall the infamous case of the DAO (Decentralized Autonomous Organization) incident [5], where a vulnerability in a smart contract led to the loss of millions of dollars worth of cryptocurrency, highlighting the critical importance of robust security verification and testing activities in smart contract development. The potential presence of vulnerabilities in the smart contract code gets even more relevant when considering that once deployed, a smart contract is hard to patch (due to the immutability property of the underlying blockchain).

Let us note that the first step toward the design and implementation of secure software is to consider security-by-design principles in the development process and to support security testing. To this aim, a key enabling factor is the knowledge of the potential security vulnerabilities that need to be detected and resolved before the software becomes operational.

When dealing with traditional software, there is a consolidated process and well-established naming conventions to report, classify and document discovered vulnerabilities, contributing to an increased level of awareness among developers and security testers. This is currently enabled by (i) the definition of standardized formats such as the *Common Vulnerability and Exposure* (CVE) [48] and the *Common Weaknesses Enumeration* (CWE) [49] to report and classify vulnerabilities and weaknesses and (ii) the maintenance of repositories and databases such as the NIST NVD [57] and the CWE repository.

While smart contracts share similarities with traditional software systems, the inherent structural distinctions between the two give rise to unique issues requiring specialized attention. Challenges such as *reentrancy* attacks [5] and *gas limit* vulnerabilities [47] are rooted in blockchain's unique features, including decentralized execution, and may go beyond traditional coding flaws. These examples suggest that the existing vulnerability classification schemes are not enough to fully capture and represent issues affecting smart contracts and leave space for ambiguity and/or multiple definitions of the same issues. In addition, traditional software vulnerability and weaknesses taxonomies may often fall short when it comes to smart contracts [80], due to the structural and conceptual differences between traditional software and smart contracts. Indeed, they are either too general or too specialized towards traditional platforms, failing to adequately capture the nuances of the blockchains and their distributed execution paradigm. Thus, not surprisingly, in the last two decades, the research and industrial community has made a large number of proposals [5, 18, 34, 40, 44, 46, 51, 65, 68, 81–83, 93] aimed at classifying the vulnerabilities affecting smart contracts from different perspectives. Despite all these efforts, there is no yet a *standard de facto* and a shared naming convention to identify and classify issues discovered during the smart contract testing phase and there still exists a gap between the precise and structured side looking to traditional software and the chaotic one looking to smart contracts, making it very hard in practice to reason on smart contract vulnerabilities, e.g., to compare different vulnerability detection tools or to carry out any study in the blockchain ecosystem. Just to give an additional example highlighting this gap, let us note the difference in the granularity of the taxonomies and classifications currently existing for traditional software and smart contracts. Indeed, according to the NIST definition, a software vulnerability [58] refers to a flaw within a system that malicious entities can exploit to compromise its integrity, confidentiality, or availability. In contrast, a weakness [59] represents a sub-optimal aspect in the design or implementation of software, holding the potential to lead to vulnerabilities. When looking at the smart contract world, the distinction between these two concepts becomes subtle and the terms vulnerability, weakness, issue and defect are often used as synonyms[2].

Contributions. Given the vast amount of attempts and proposals available in the literature to classify and categorize smart contracts issues, this paper aims to critically review the state of the art on smart contract vulnerability taxonomies to identify their similarities and differences, to quantify their relative level of coverage and to highlight the relevant features that should be included in every vulnerability classification scheme. A key result of our review is that existing proposals lack a proper formalization, making them impractical for several use cases and hard to maintain even for a motivated community. Hence, we propose a novel data model that can capture the nuances of smart contract vulnerabilities, possibly supporting the community in different use cases. In particular:

- We review a large number of earlier studies aimed at smart contract vulnerability taxonomies, pinpointing their main characteristics and limitations;
- We perform a systematic comparison of such taxonomies, identifying overlapping terminologies and common traits;
- We propose a novel data model whose aim is to provide a more structured view of smart contract vulnerabilities by clearly defining the key entities and relationships relevant to the blockchain ecosystem;
- We show that our data model can naturally fit existing proposals and then provide a preliminary instantiation able to organically put together different taxonomies;
- We demonstrate that a taxonomy built on a well-structured data model can effectively support several relevant use cases, possibly helping the blockchain community to make smart contracts safer.

Release of our proposal. The data model and its current instantiation are available in the companion project [70]. Our platform [71] allows users to interactively explore our data model instance, dump the underlying raw data, and update it via push requests.

Structure of the paper. The paper is structured as follows. Section 2 presents the existing smart contract taxonomies, comparing them and identifying their limitations. Section 3 describes our data model and puts it in perspective with existing works. Section 4 explains how our data model can support different use cases. Finally, Section 5 provides the final remarks, highlighting possible improvements to our proposal.

2 COMPARATIVE ANALYSIS OF EXISTING SMART CONTRACT VULNERABILITIES TAXONOMIES

This section reports the results of our literature review focused on the identification, comparison and analysis of the existing taxonomies used to report and classify smart contract vulnerabilities.

[2]We will analyse this aspect in depth in Section 2.

The review considered a substantial volume of literature dealing with smart contract vulnerabilities from different perspectives, including even studies related to vulnerability detection tools. Our analysis has been driven by the following research questions:

- **RQ1:** Which are the existing taxonomies of smart contract weaknesses or vulnerabilities and how do they classify the flaws?
- **RQ2:** Are these taxonomies complete and comparable in terms of categorization criteria and adopted nomenclature?
- **RQ3:** Is there any taxonomy that is seen as the *standard de facto* by the community? If not, what are the main reasons that may prevent the adoption of existing proposals?

Answering these questions would give us a clear picture of the level of standardization and interoperability of existing smart contract vulnerability taxonomies and would shed light on the gap existing between traditional software vulnerability classification approaches and those currently in place when dealing with smart contracts, pointing out future research directions.

Review Methodology. We identified the final set of papers considered in the review by following a selection process aimed at collecting, filtering and finally grouping papers relevant to our research objectives. Initially, we searched for studies on the main digital libraries such as Google Scholar, IEEE Xplore and ACM Digital Library using a predefined meta-query to filter articles. Our meta-query combined the keywords *smart contract, weaknesses, vulnerabilities, languages, tools,* and *taxonomies* as follows:

```
smart contract AND (weaknesses OR vulnerabilities
    OR languages OR tools OR taxonomy)
```

This preliminary search focused on papers related to the macro areas of smart contracts languages, taxonomies and tools. It was aimed to investigate the existence of exhaustive taxonomies and any mappings of detection tools, the presence of comparative analyses of the efficacy of different tools, and to account for weaknesses and vulnerabilities defined for different smart contract languages and blockchains. The process led to an initial list of 88 papers and open-source projects available in Table 3 of our technical report [74].

In the second stage, it became apparent to us that the vast majority of works focused on Ethereum and its programming language Solidity. Hence, to avoid considering too heterogeneous proposals and exotic technologies, we restricted our review to these two main technologies. Also, we noticed that most of the studies presenting tools frequently list the detected vulnerabilities but a proper taxonomy is often omitted. Based on these aspects, we defined a set of exclusion/inclusion criteria to filter the initial list of works:
Inclusion criteria:

- **X:** Papers containing an extensive, comprehensive taxonomy and possibly an efficacy analysis of tools;
- **Y:** Papers with detailed vulnerabilities or weaknesses descriptions, either enabling clear distinctions among different variations of the same issue or providing additional insights about attacks and defences;
- **Z:** Papers adhering to criteria X or Y should be either recent, e.g., published after 2020, or highly cited by smart contracts systematic literature reviews (SLRs) or other articles discussing relevant taxonomies;
- **V:** Papers mapping smart contracts weaknesses or vulnerabilities to well-known reference schemes, such as DASP [34], SWC [81], or CWE [49];
- **W:** Papers that considerably vary in the type of issues addressed to maximize vulnerability and weakness coverage, as well as in the criteria used for categorization (e.g. root causes, layers, security properties violations, severity, CWE classes, defect consequences, etc.).

Exclusion criteria:

- **J:** Papers not tackling the Ethereum blockchain or Solidity programming language;
- **K:** Papers on tools not providing exhaustive descriptions of vulnerabilities or mappings of the covered vulnerabilities.

Comparing works against the defined criteria, we narrowed down the initial list to 57 publications. Subsequently, we further refined it by including a wide range of open-source project repositories, which were not necessarily associated with scientific publications, provided they were presenting taxonomies and were highly cited or used as references by other works, in compliance with criterion *Z*. The resulting list of 77 elements was then evaluated by two of the authors to prune away non-relevant contributions, obtaining a final selection of 63 works, available in Table 2 from our technical report [74].

RQ1: Discussion of the Existing Taxonomies. The analysis of selected works reveals that there have been several prior efforts aiming at producing a unified taxonomy for smart contract weaknesses and vulnerabilities. The rationale behind this is that the availability of a common reference for smart contract defects may provide valuable support for detection tools by enabling the mapping of vulnerable code on well-defined bugs and flaws, and in parallel assist developers in the writing phase by increasing their overall level of understanding and awareness.

A first interesting observation looking at the vocabulary used in the analysed proposals is that the blockchain community seems to interchangeably use the terms *vulnerability* and *weakness*, without making a strong distinction between the two and favouring overall the term *vulnerability*. Hence, any weakness is often considered a vulnerability, even when no security implications have been proved[3]. As underscored by several works [1, 4, 5, 17, 32, 45, 84, 96], this likely stems from the immutable nature of the blockchain, which makes it hard to *upgrade* a deployed smart contract, and the critical domains where smart contracts are playing a role, e.g., Decentralized Finance, justifying a conservative evaluation against any weakness. This naming convention is currently in contrast with the vocabulary used to identify issues in traditional software where NIST provided clear definitions for the concepts of vulnerability, weakness, bug etc. However, to remain consistent and comparable with the existing nomenclature used in smart contracts, from now on we only use the term *vulnerability* even if the identified issue is closer to a weakness.

[3]A statistical analysis of the terminology adopted to refer to the concept of *weakness* in smart contracts found that the term *'vulnerability'*, to denote potential security threats, is employed by almost 70% of the literature, while a minority uses *'bug'*, *'issue'*, *'weakness'*, or *'defect'*. The analysis was conducted on the complete set of works available in our technical report [74].

A second point is that commonly adopted vulnerability classification schemes include the Decentralized Application Security Project (or DASP) Top10 [34], the multi-level taxonomy presented by Atzei et al. [5] and the SWC registry [81], which represents a first attempt to standardize defects IDs, aligned also with the Common Weaknesses Enumeration (CWE). However, the CWE [49] looks more suitable for traditional software than for the smart contracts context due to its higher level of abstraction. For example, while CWE might categorize a vulnerability as *CWE 710 - Improper Adherence to Coding Standards*, schemes like SWC offer more precise classifications tailored to smart contract vulnerabilities. For instance, SWC-100 specifically addresses issues like improperly set function visibility in Solidity, providing developers with more actionable guidance. Unfortunately, SWC and DASP Top10, are largely incomplete and unmaintained as they have not been updated for the last 3 years.

More recent works have proposed extensive vulnerability taxonomies [11, 44, 52, 68, 84, 93]. Vidal et al. [93] present an extended hierarchical taxonomy grouping defects in categories and subcategories. They ascribe specific names to issues and define mappings to SWC, CWE and correspondent names in other references, however, their nomenclature often does not align with that adopted by other studies. The work from Rameder [68] organizes vulnerabilities in 10 classes, synthesizing and mapping information from the taxonomies proposed by DASP, SWC, and 17 earlier surveys. Kushwaha et al. [44] categorize issues based on their root and subcauses, and define a nomenclature accordingly. Additionally, they map issues to attack instances, preventive methods and detection tools. The subdivision into levels (Solidity, EVM and Blockchain) is adopted by several works [5, 16, 23, 67, 97]. Chen et al. [11] define an extensive list of vulnerabilities at four main architectural layers (*Application, Data, Consensus*, and *Network*, respectively) providing detailed descriptions of issues and real-world attacks and insights into causes, attacks consequences and defences.

In Munir and Taha [52] safety and liveness properties are used to group vulnerabilities but their approach lacks naming standardization, while Grishchenko et al. [33] defines issues in terms of security properties violations. Dingman et al. [25] leverage the NIST Bugs Framework [56] to classify bugs, adopting the same arrangement in security, functional, developmental and operational categories identified by the previous work from Tikhomirov et al. [88].

In a recent study, Zhang et al.[96] analyzed 516 exploitable bugs found in 167 real-world contracts reported or exploited in 2021-2022, sourced from Code4rena contests and real-world exploit reports. The authors classify functional bugs in the two main categories of *Machine Auditable Bugs(MABs)* and *Machine Unauditable Bugs(MUBs)*, defining further subcategories for each. A recent work from Chaliasos et al. [10] considers vulnerabilities at the *Smart Contract* and *DeFi Protocol* layers, while Kalra et al. [42] distinguish among correctness, fairness and mining-derived issues.

All the foregoing works primarily focus on classifying defects and encompass the concept of vulnerability categories, distinguishing among different spheres or domains. Nevertheless, none of them offers a fully exhaustive coverage in terms of defects descriptions, examples, defences and mappings to attacks, detection tools, or previous works and classifications all at once. Furthermore, they do not often examine possible relationships among defects. The concept of vulnerability propagation and relations is faced by Staderini et al. [84], who define *subset* and *implication* relationships and vulnerabilities *intersections*, mapping issues to CWE weaknesses. The studies conducted by Praitheeshan et al. [64] and Qian et al. [67] link Solidity vulnerabilities to real-world attack instances and relevant security issues. Only a handful of studies [51, 84, 85, 93, 97] attempt to correlate defects with CWE classes.

Other significant contributions to smart contracts vulnerability classifications come from several community-oriented projects [20, 29, 38, 40, 40, 51, 54, 65, 77–79, 82], or tools repositories [83]. A further drawback of existing classification schemes is that they often do not accurately reflect the current state of practice, as pointed out by Vidal et al. [93]. Among the aforementioned projects, only the EthTrust Security Levels specification from the Enterprise Ethereum Alliance [54] and the Smart Contract Security Field Guide for Hackers [51] are actively maintained.

Short answer to RQ1. There exists a large body of works proposing smart contract vulnerability taxonomies. These proposals classify the flaws according to different dimensions, such as *levels*, *categories*, *layers*, *root causes*, and traditional security properties. This extremely fragmented scenario makes it unclear what is the relationship and *compatibility* among such taxonomies and whether there is a natural evolution among them.

RQ2: Comparative Analysis of Existing Taxonomies. To evaluate how related and *compatible* are the existing taxonomies, we have performed a fine-grained comparative study. The undertaken analysis primarily focused on evaluating the overlap among taxonomies, their degree of exhaustiveness, the extent to which categorization criteria differ, the usage of a standard nomenclature and the correspondences established on common reference schemes.

Apart from the overlapping and nomenclature aspects, a subset of studies has been considered to analyze the other dimensions. Studies are picked by taking into account the completeness of provided taxonomies in terms of some predefined *quality attributes*, including the number of addressed vulnerabilities, presence of clear descriptions for bugs and flaws enhanced with explanatory examples, and possible categorization according to levels or other criteria. Furthermore, we assessed the existence of mappings between vulnerabilities and real-world exploits, tools suitable for their detection, potential countermeasures and other works handling the issue. Another aspect examined is the presence of mappings to IDs in the SWC Registry [81] and CWE [49].

Table 1 summarizes the findings, presenting a subset of the analyzed attributes for each of the studies. For the sake of space, Table 1 only lists the 36 works that, directly or indirectly, propose a relevant taxonomy. As seen in the column *Classification criteria*, the majority of the works define categories of diversified nature to group similar issues. However, as reported by column *Level*, a subset of them at least share the concept of *level*. In general, we can observe that many studies tend to have a coarse-grained granularity, with few exceptions [11, 15, 25, 93, 96]; this may not fully meet the requirements of detection tools, which require higher level of detail and whose progress state is often more advanced. Despite that,

Table 1: Relevant works presenting smart contract vulnerabilities taxonomies.

Reference	Classification criteria	Levels	# of vulnerabilities	CWE mapping	SWC mapping
Kushwaha et al. 2022 [44]	Root/Sub causes	✓(Root causes)	23	✗	✗
Vidal et al. 2023 [93]	Categories/Subcategories	✗	76	✓	✓
Rameder 2021 [68]	Classes	✗	54	✗	✓
Chen et al. 2020 [11]	Ethereum Architectural Layers, Causes	✗	40	✗	✗
Dingman et al. 2019 [25]	NIST Bugs Framework, Categories (Issue types)	✗	49	✗	✗
Staderini et al. 2020 [84]	CWE classes	✗	28	✓	✗
Staderini et al. 2022 [85]	CWE classes	✗	33	✓	✗
Munir and Taha 2023 [52]	Safety/Liveness Properties	✗	61	✗	✗
Atzei et al. 2017 [5]	Levels	✓(Levels)	12	✗	✗
Huang et al. 2019 [37]	Causes, Weaknesses	✗	10	✗	✗
Zhou et al. 2022 [97]	Levels	✓(Levels)	13	✓	✗
Praitheeshan et al. 2020 [64]	Software Security Issues	✗	16	✗	✗
Chen et al. 2022b [14]	Categories (Consequences)	✗	20	✗	✗
Chaliasos et al. 2023 [10]	Layers	✗	28	✗	✗
Kalra et al. 2018 [42]	Correctness/Fairness Issues	✗	10	✗	✗
Chen et al. 2017 [15]	Categories (Gas-costly patterns)	✗	7	✗	✗
Grishchenko et al. 2018b [32]	Security Properties Violations	✗	8	✗	✗
Tsankov et al. 2018 [91]	-	✗	11	✗	✗
Tikhomirov et al. 2018 [88]	Issue Types	✗	21	✗	✗
Qian et al. 2022 [67]	Levels, Security Issues	✓(Levels)	15	✗	✗
Hu et al. 2023 [36]	Defect Types	✗	20	✗	✗
Chu et al. 2023 [16]	Threat Levels	✓(Threat Levels)	12	✗	✗
Amiet 2021 [3]	Blockchain Ecosystem Components	✗	12	✗	✗
Consensys Diligence Consensys [18]	8	✗	-	✗	✓
di Angelo and Salzer 2019 [23]	Categories	✓(Categories)	16	✗	✗
SC Security Field Guide for Hackers 2023 [51]	Categories	✗	27	✓	✗
SigmaPrime SIGP Prime [65]	-	✗	16	✗	✗
DASP Top10 Group [34]	-	✗	10	✗	✗
Securify 2.0 SRI Lab [83]	Severity	✗	37	✗	✓
SMARTDEC Classification SmartDec [82]	Classes/Groups	✗	33	✗	✓
Trail of Bits (Not So) Smart ContractsCrytic [20]	-	✗	11	✗	✗
Luu et al. 2016 [46]	-	✗	4	✗	✗
Brent et al. 2018 [8]	-	✗	5	✗	✗
Grech et al. 2018 [31]	-	✗	3	✗	✗
Bose et al. 2021 [6]	-	✗	2	✗	✗
Zhang et al. 2023 [96]	Categories (Machine Auditable/Unauditable)	✗	39	✗	✗

studies that focus on tools tend to narrow their scope to issues identified by the specific tool being presented. Although some works offer comprehensive defect lists (see column *# of vulnerabilities)* and succeed in providing most of the identified quality attributes, none of the references can be considered complete in all respects. Additionally, to the best of our understanding none of the studies considers vulnerabilities interdependencies, exception made for Staderini et al.. We can also observe that most of the references fail to provide associations to established schemes like CWE or SWC. Additionally, major references and open-source taxonomies either lack maintenance or receive limited community participation in updating online repositories, as highlighted by Vidal et al. [93].

To further examine the overlap between different taxonomies and quantify their level of overlap and/or complementarity, a vulnerability mapping process has been conducted across 57 out of the 63 works[4] to group defects with similar semantics under unified names, assessing their analogies and common traits based on the specified description. The mapping process was hindered by the heterogeneity of nomenclatures and classification criteria and required to accurately compare descriptions to avoid ambiguity. This sets the stage for a subsequent examination of the semantic and syntactic similarities among works. Specifically, the former involves investigating the number of common vulnerabilities (i.e., issues semantically equivalent but identified with different names) while the latter delves into the exact syntactic correspondence of the employed nomenclatures (i.e., same issues named in the same way across different proposals), thereby verifying to which extent the schemes adhere to standard naming conventions. The semantic and syntactic similarities among a pair of works can be computed using the Jaccard [95] coefficient, which can measure the ratio of the intersection and union of their vulnerability sets.

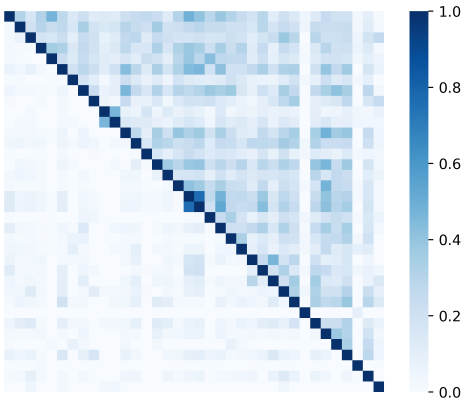

Figure 1: Heatmap showing the syntactic (bottom left) and semantic (top right) similarity across taxonomies. The works are considered on the two axes in the same order as in Table 1.

The results of our similarity analysis for the works in Table 1 are visually summarized by the heatmap in Figure 1, while the full set of values can be retrieved from the table of similarities in our repository [70]. The bottom left side of the heatmap represents the

[4]We excluded from the analysis community projects without a taxonomy freely available in a repository.

syntactic similarity across taxonomies, which have been considered in the same order as in Table 1, while the top right side shows instead the semantic similarity. As expected, the maximum correlation (dark blue) on the diagonal results from the perfect correlation of each work with itself. The presence of very low values (light blue) for syntactic similarities highlights a significant heterogeneity between vulnerability names, which suggests the absence of a uniform naming convention. Even when considering the semantic similarity, the situation does not improve significantly, as shown by the values which struggle to exceed 0.5, indicating a weak overlap among taxonomies. This can be influenced by multiple factors, including the type of defects considered, their granularity and the range of vulnerabilities covered. Higher semantic similarity values only occur in cases involving works by the same authors, references presenting a tool and its repository, or studies declaring to explicitly reference a taxonomy from another study. Even in these instances, similarity in nomenclature remains low.

Short answer to RQ2. Our comparative analysis reveals that taxonomies generally lack coherence, with no standard nomenclature being commonly adopted. They vary significantly in expressive power, coverage, class types, and categorization criteria. Consequently, the comparison process is complex and direct translation between taxonomies is not straightforward. Furthermore, they rarely provide mappings to IDs of common reference schemes and cannot be regarded as entirely exhaustive in providing descriptions, examples, related concepts such as attacks, mitigations, tools and ultimately, coverage and granularity.

RQ3: Adoption of Taxonomies from the Community. Even if there is a large number of taxonomies and they appear to be quite incompatible with each other, one natural question is whether the blockchain community is gradually moving towards one specific vulnerability taxonomy, hence, solving in practice the fragmented scenario depicted by the previous sections. We have thus analyzed how the existing taxonomies are used in the wild by the community, especially when considering the state-of-the-art analysis techniques for smart contracts [7, 8, 19, 21, 28, 30, 31, 42, 43, 46, 50, 55, 62, 66, 76, 88–90, 92] and parallel studies on real-world attacks [5, 11, 24, 65]. Unfortunately, we observed that no taxonomy can be seen as the *standard de facto*. More interestingly, we attempted to evaluate the reasons why existing taxonomies are not widely adopted by the community. A predominant motivation is the lack of coverage. Unfortunately, no easy solution can be likely offered to tackle such a problem due to the ever-evolving nature of smart contract vulnerabilities. However, we noticed that, e.g., works presenting tools rarely decide to extend existing taxonomies but rather often propose novel vulnerabilities under incompatible classification schemes. Additionally, the majority of tools tend to produce outputs that do not refer to existing taxonomies, even when such tools consider vulnerabilities that are already well-known by the community. Nonetheless, we believe that this is not surprising. Indeed, existing taxonomies are not designed for practical use within tools due to their lack of formal structure beyond basic descriptions. The key concepts around vulnerabilities and their relationships have not been properly formalized. Finally, the existing schemes are quite rigid and can hardly be updated. For instance, Vidal et al. [93] have recently proposed OpenSCV, an open hierarchical taxonomy for smart contract vulnerabilities. However, their scheme groups vulnerabilities in a tree structure, without offering the functionality to define the *subset* and *implication* relationships proposed by Staderini et al. [84]. Additionally, the associated GitHub project does not provide the taxonomy in a well-structured format, which may hinder its usage within tools and make it impractical to maintain over time.

Short answer to RQ3. None of the existing taxonomies can be considered as the *standard the facto*. Besides coverage issues, a major aspect that seems to limit the adoption of existing taxonomies is the lack of a formal structure beyond basic descriptions, which makes them hard to use, e.g., within smart contracts analysis tools, and quite hard to improve over time.

In the rest of this article, we attempt to propose a novel data model that can capture the relationships among smart contract vulnerabilities and other entities, meeting the needs highlighted by prior works proposing taxonomies. However, differently from such works, our data model can support the construction of a new taxonomy that is properly structured, can be effectively stored in different formats, favouring, e.g., adoption by smart contract analysis tools, and possibly easily updated over time by the community.

3 DATA MODEL FOR SMART CONTRACT VULNERABILITY TAXONOMIES

In this section, we present our data model for smart contract vulnerabilities. We first explain what the data model is expected to provide, then we present the key aspects behind it, considering also a preliminary instantiation of the model, and, finally, we discuss how it can be compatible with most existing taxonomies.

3.1 Design Principles

After reviewing a large number of smart contract vulnerabilities taxonomies, we identified the lack of formalization as a common and crucial limitation. However, this limitation actually brings several implications that inherently may harm the adoption of a taxonomy in the community. To take into account this limitation and its implications, we have designed our data model and then worked on its preliminary instantiation, driven by a few key principles:

- *Structured.* The data model should clearly define which are the key entities and their relationships in the context of smart contract vulnerabilities. While such formalization could be performed through different formal schemes, we believe that an entity-relationship (ER) model is the most natural solution as it is likely the most popular modelling approach in computer science.
- *Expressive.* The data model should meet the needs of the community by integrating the key concepts and main relations that emerged in previous existing taxonomies. Nonetheless, it should critically combine such proposals to avoid uncontrolled complexity and excessive specialization. Hence, our data model should try to be *compatible* with the literature and relate to well-known reference schemes.

- *Usable.* The data model should be defined considering different kinds of users. First, as done by existing taxonomies, the model should be *intuitive* for human users, such as developers, security auditors, and researchers. This implies that it should contain several attributes that can offer, e.g., intuitive descriptions of specific concepts tracked by the model. Second, the model should be practical for exploitation by automatic tools, e.g., vulnerability detection tools. This implies that, e.g., some relationships should be defined to facilitate the parsing and evaluation within automatic frameworks. Moreover, the data model instantiation should be available in *easy-to-process* data formats, e.g., JSON files.
- *Extensible.* The data model and its instantiation should be perceived as a moving target requiring a continuous improvement cycle given the ever-evolving blockchain scenario. Hence, any concrete output associated with the model should be made available to the community in a way that favours its update over time, while requiring minimal effort. This implies that the data model should be envisioned within a collaborative project open to the community.

In the next sections, we describe how our model and its instantiation pursue these principles. Moreover, Section 4 discusses and partially demonstrates several use cases enabled by our proposal.

3.2 Data Model Entities and Relationships

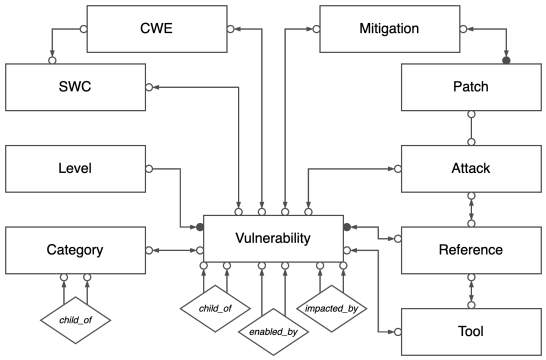

Figure 2: ER data model for smart contract vulnerabilities, represented according to the Bachman notation.

The proposal for a data model able to describe smart contract vulnerabilities is visually represented at a high level in Figure 2. While the proposal has been formally defined as an entity-relationship (ER) model, for the sake of the discussion, we informally present it in this section. To determine relevant attributes, entities and relationships we started by identifying the key attributes used in existing taxonomies. We upgraded them to entities whether a more detailed specification was needed, specifying corresponding relationships with paired vulnerabilities. In the centre of Figure 2 we can find the entity *Vulnerability*, which, as expected, is the core of the entire model. Such entity is then surrounded by nine other entities (*Level*, *Category*, *CWE*, *SWC*, *Tool*, *Attack*, *Mitigation*, *Patch*, and *Reference*), possibly linked together through different relationships. Each entity and relationship is now described and motivated in detail.

A *vulnerability* is a defect which may affect a smart contract and bring security consequences. Consistently with prior works [1, 4, 5, 17, 32, 45, 84, 96], such term is abused to cover any weakness that may have a security implication. Differently, the term *attack* will be used to refer to a DApp-specific vulnerability instance.

A *vulnerability* is characterized by four key attributes[5]: a unique *id*entifier, a *name*, a *description*, and a *sample code*. The latter should exemplify the essence of the problem, helping to understand what the description is informally asserting. As described in Section 2, the community often uses different names for the same vulnerability. Moreover, a vulnerability may have been discussed in detail by different sources. To cope with these observations, the entity *reference* representing bibliographic resources is introduced and linked to *vulnerability*. Such a relationship has the optional attribute *alias* to report the specific name proposed by each reference.

A *vulnerability* can be categorized according to different and orthogonal dimensions. On one side, the vulnerability may arise from a specific conceptual *level* related to a smart contract. For instance, consistently with prior works [5, 24, 39, 67, 87, 97], at least three major conceptual *levels* can be identified: the Solidity Language, which includes issues resulting from the nuances of the high-level language used to write the smart contract; the EVM, which encompasses any issue resulting from the low-level execution of the smart contract in a single node; and the Ethereum Blockchain, which comprises problems due to the distributed execution paradigm exploited by a smart contract. On the other side, *vulnerabilities* can be grouped according to one or more *categories*, where a *category* represents a broad flaw class under which different vulnerabilities with a similar nature can be grouped. For instance, *integer overflow* and *improper rounding* can both fall under the *arithmetic category*. As pointed out in previous works, categories can be mapped to a hierarchical view, where higher-level categories are parent, i.e., more general than lower-level categories. Moreover, the same vulnerability can be included in different categories, e.g., a flaw can both be an arithmetic issue and a business logic problem.

Vulnerabilities can often be related to each other. Three relationships may be defined: *child_of*, which links a *vulnerability* to a more generic one; *enabled_by*, which depicts whether a *vulnerability* may be caused by another one; *impacted_by*, which asserts that a *vulnerability* may be dangerous when co-existing with others[6].

The data model is quite flexible when aiming at organizing vulnerabilities in a hierarchical structure. As seen in previous works, some taxonomies prefer to define several categories and then devise a hierarchy among them (exploiting the *child_of* relationship among *categories*), while other taxonomies define the hierarchy directly over the *vulnerabilities* (exploiting the *child_of* relationship among *vulnerabilities*). Since both approaches are reasonable, both of them are supported in our model.

Vulnerabilities affecting smart contracts may be related to well-known flaws in the context of traditional software. In particular, CWE is an effective weakness classification framework that contains instances relevant also in the smart contract's ecosystem.

[5]For the sake of space, attributes are hidden in Figure 2 but a concrete example is provided in Figure 3.
[6]The community often labels some weaknesses as vulnerabilities. The *impacted_by* link thus exemplifies that a vulnerability is dangerous when other conditions are met.

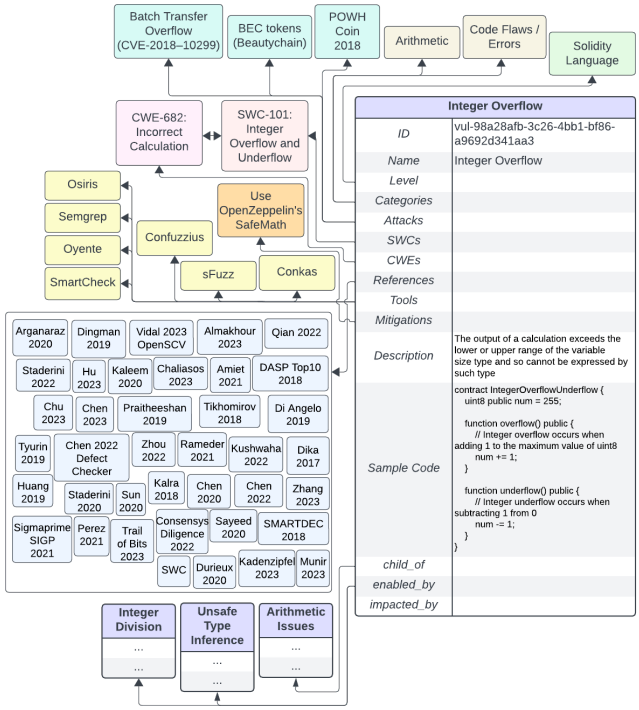

Figure 3: Example: instantiation of the vulnerability *Integer Overflow* according to our data model.

Hence, an entity *CWE* representing existing CWE instances is defined and a relationship between *CWE* and *vulnerability* is established. Similarly, when considering existing classification schemes tailored to the blockchain ecosystem, SWC is considered. This is a taxonomy often mentioned in the literature [2, 12, 12, 26, 36, 68, 69, 84, 85, 93], and exploited by several existing detection tools [19, 83, 90]. Unfortunately, SWC is nowadays unmaintained and incomplete. Nonetheless, given its historical relevance, an entity *SWC* is included and linked with *vulnerability*. Finally, *SWC* may be conceptually mapped to *CWE*, thus we optionally link them.

A vulnerability may be detectable by some tools, so the entity *tool* is defined and linked to one or more *vulnerability*. A tool can have different key attributes, such as a *name*, a *type*, a unique *id*entifier, and the *year* when it was published. Each tool may be available on a specific website and may have been presented in one or more papers, hence *tool* may link one or more *references*.

A vulnerability may have been exploited in DApp-specific *attacks*. While most attacks in the blockchain ecosystem are not yet properly tracked and documented in public repositories, such as CVE, the most interesting ones are often the subject of extensive studies and used as motivations in several works. Our data model hence devises an entity *attack*, that is mainly intended to represent well-known attacks linked to one or more *vulnerability*. Each attack has several attributes, such as a *name*, a unique *id*entifier, and the *year* of exploitation. The attack may have been discussed in detail in several bibliographic *references*, hence a relationship among these two entities exists. Finally, an *attack* may have been fixed with a DApp-specific *patch*. The *patch* may have been developed according to some *mitigation* best practices, hence the entity *mitigation* is introduced and linked to one or more *patches*. Internally, the *patch* contains an attribute pointing to the code fixes applied by the DApp developers. Since a mitigation is often suggested based on the vulnerability it addresses, a link between *mitigation* and *vulnerability* is established. This relationship reflects the best practices to handle (or avoid) a vulnerability.

3.3 Instantiating the Data Model

The proposed data model is an interesting conceptual framework that, however, has limited practical value for the end users. For this reason, we have decided to start instantiating it with the goal of devising a new smart contract vulnerability taxonomy. In this section, we first present a sample of such instance, since this may help to clear out the role of each entity in the data model. Then, we explain how we plan to turn this first preliminary instantiation into a richer taxonomy through a community-driven effort.

Example. Figure 3 considers the vulnerability *integer overflow*. This flaw is well-known even in the context of traditional software and usually emerges when an integer value, due to, e.g., an arithmetic addition, *wraps* around, i.e., turns from a large value into a low value. Interestingly, this issue can arise even in smart contracts and it is regarded by the community as a critical flaw possibly having strong security implications. Our taxonomy provides a description of the problem and a piece of code to exemplify its essence.

Integer overflows in smart contract may be often caused, i.e., *enabled*, by more high-level vulnerabilities, such as *Integer Division* or *Unsafe Type Inference*, as shown by Figure 3.

Users may naturally expect to find the vulnerability *integer overflow* under the category *incorrect arithmetic operations*, thus a category for this purpose was defined. However, at the same time, some tools that can detect *integer overflow* categorize it using the term *arithmetic issue*. To favour mapping between tools' output and the taxonomy, the higher-level vulnerability *arithmetic issue*, parent of *integer overflow*, was also defined. Prior works also categorized the vulnerability *integer overflow* under the broad group named *Code Flaws / Errors*, hence, a link between these two was defined.

Prior works seem to agree that the vulnerability *integer overflow* in Ethereum arises due to the nuances of the Solidity programming language [90]. Indeed, while it is expected that the underlying EVM can only deal with fixed-size data types, it may at the same time be pretended as a viable strategy from the higher-level programming language to avoid or deal with integer overflows. This is similar to the traditional software scenario where it is well-known that the underlying platform, e.g., Intel x86, only works with fixed-size numbers, but taking benefit from arbitrary-precision integers when using high-level programming languages is expected. Hence, since Solidity does not come with arbitrary-precision integers, a link between *Solidity Language* and *integer overflow* was defined.

The vulnerability *integer overflow* has been discussed in detail by several references [2–4, 10–14, 16, 18, 20, 23–26, 34, 35, 37, 40–42, 44, 52, 63–65, 67, 68, 75, 81, 82, 84–86, 88, 93, 97], which were thus linked to *integer overflow*. Similarly, several analysis tools [21, 46, 55, 88–90, 92] are able to potentially detect this vulnerability and thus were linked in the taxonomy, as shown in Figure 3.

Since *integer overflow* is a well-known flaw even in traditional software, it can be easily linked to *CWE-682: Incorrect Calculation*. Moreover, SWC also recognizes this flaw as a common issue even in the blockchain ecosystem, assigning it to the identification number *SWC-101: Integer Overflow and Underflow*. Links from *SWC-101* to *integer overflow* and from *SWC-101* to *CWE-682* were thus created.

Integer overflow has been at the core of at least three attacks known as *Batch Transfer Overflow* (publicly tracked by *CVE-2018-10299*), *POWH Coin 2018*, and *BEC Tokens (Beautychain)*. These attacks have been patched in different ways, including through the usage of OpenZeppelin's SafeMath [60, 61], a common mitigation best practice for *integer overflow* in Ethereum.

For the sake of simplicity, Figure 3 mainly shows relationships associated with *integer overflow* and does not fully demonstrate the value of multi-level hierarchical relationships. Figure 4 is thus more adequate for such goal and considers the vulnerability *Balance Equality*. Such flaw is quite dangerous when other vulnerable conditions are verified, including *Ether Transfer with Selfdestruct*, *Ether Transfer with Mining*, and *Pre-sent Ether*. Interestingly, *Ether Transfer with Selfdestruct* in turn can be caused by *Unprotected Selfdestruct*. Finally, *Balance Equality* can be seen as a specialization, e.g., child, of *Requirement Violation*, which in turn is child of *Assert / Require / Revert Violation*.

Collaborative project. The instantiation of the data model is not yet fully accurate and complete. Unfortunately, reaching such a state is extremely ambitious and quite impractical for a limited set of contributors given the large number of ever-evolving vulnerabilities, tools, mitigations, and attacks. Hence, while on the one hand, we decided to invest a significant amount of effort to bootstrap the taxonomy, on the other hand, we have chosen to also design a collaborative platform where the community could cooperate to reach such an ambitious goal, possibly allowing the taxonomy to outlive its original creators and initial release.

In more detail, improving our taxonomy merely requires making push requests on a GitHub project. Indeed, to make trivial the update of the taxonomy, it is stored as a JSON file. Users can then follow a straightforward 3-step workflow:

(1) fork the project on GitHub;
(2) edit the JSON file via a local editor or the GitHub editor;
(3) generate a pull request to the upstream repository.

Upon acceptance of a pull request, a GitHub action verifies the integrity of the JSON database, reverting the update if it doesn't comply with the defined schema[7]. Otherwise, a separate action regenerates any derived representation of the data model instance and re-deploys the interactive web platform (see Section 4). Finally, GitHub issues could be used to report inconsistencies and start a discussion about major structural improvements.

3.4 Retrofitting Existing Taxonomies

In this section, we show that our data model is well-suited to capture a significant portion of the existing taxonomies. To this end, we consider several relevant taxonomies, selected from recent or highly cited sources, favouring the ones including various levels of structuring and exposing diversified layering densities.

[7]The JSON schema can be found at [72].

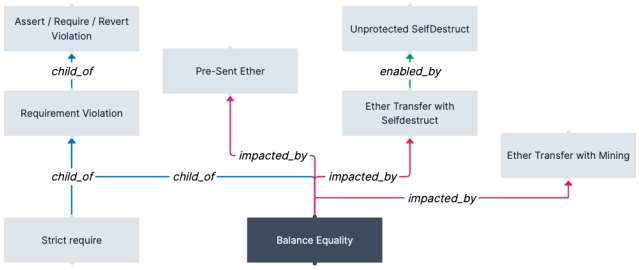

Figure 4: Example: sample of the relationships related to the vulnerability *Balance Equality*.

A prime illustration is the taxonomy presented by Kushwaha et al. [44], cited by several recent works [16, 52, 97]. Here the root causes can be straightforwardly regarded as the *levels* and the sub-causes as *categories* with 1:1 associations. Alternatively, to explicitly denote the association between sub and root causes, both can be modelled as *categories*, specifying their hierarchical relationship through *child_of* relations. It is even feasible to represent detection tools, attacks, and prevention mechanisms as entities within the data model: as a result, a *vulnerability* will be linked to a *category*, a list of *tools*, an *attack*, e.g., example of real-world exploit, and an associated *mitigation* best practice, which may encompass more than one *patch* action. We can apply the same approach to other more recent works: in Munir and Taha [52] properties, platform, languages, and studies may be aligned respectively to *levels*, parent and child *categories* and *references*; while in Vidal et al. [93], aside from evident categories and subcategories, defect types and qualifiers can be specified either as *level* entities or within the *description* attribute, each associated 1:1 to vulnerabilities. Similarly, it is possible to map taxonomies with less structured categories but provide further information, as the one proposed by Rameder [68]. Alongside classes definable as *categories*, brief descriptions and works addressing a specific vulnerability can be respectively modelled using the *description* attribute and the *reference* entity. With additional effort, alternative names for a *vulnerability* can be also included in each connected reference by leveraging the *alias* attribute.

Even taxonomies that exhibit specific associations with IDs in the CWE scheme can be successfully mapped by linking *vulnerability* entities with *CWE* entities. Examples of such cases include the study by Staderini et al. [84] and the taxonomy proposed by Zhou et al. [97], where the concepts of level, CWE, and real-world attack are captured by the respective entities in the data model. Intuitively, taxonomies with a low or absent degree of categorization, such as the one highly cited in literature proposed by Atzei et al. [5], can be even more easily mapped at the discretion of the user.

Unfortunately, some taxonomies with a highly specialized structure may be harder to map into our data model, as is the case of the scheme from Chen et al. [11]. In this case, their vulnerability locations could be interpreted as *levels*, with sub-causes and causes directly modelled as *categories* with *child_of* relationships. The *attack* entity can be employed to model attack instances linked to vulnerabilities, thereby retaining their association with the location of the vulnerability they exploit. However, it remains unclear how

to adequately represent attack consequences, and the *mitigation* entity alone does not currently support the high level of hierarchical organization of their defences. This taxonomy appears highly specialized towards the aspect of mitigations, which, however, is often insufficiently documented when zooming out within the context of vulnerability detection within the blockchain community. Nevertheless, acknowledging this limitation, we are currently revising our data model to enhance its degree of hierarchical structuring for countermeasures and best practices.

Overall, we have shown that taxonomies with categorization criteria grounded in common aspects and a moderate level of organization can be instantiated into our data model.

4 USE CASES

In this section, we discuss how our data model can permit several use cases that could be relevant for the research community and, more in general, for the blockchain ecosystem. The presented use cases are split into two groups: *supported*, which align with scenarios where we already have some proof of concepts (PoCs), and *enabled*, which show potential but need additional efforts from the community before seeing any concrete implementation.

4.1 Supported Use Cases

Interactive Exploration of a Taxonomy. Our data model allows the community to express several interesting relationships across entities. However, such powerful expressiveness can easily lead to an overwhelming complexity for a human. Hence, we believe that our model should be seen as a graph, whose exploration may be supported by interactive visualization tools. Indeed, the ever-evolving nature of smart contract vulnerabilities can no longer be described through rigid and tabular listings. Hence, an interactive visualization, exploiting the well-defined structure of our data model, may help users dominate the large amount of information coming from a comprehensive taxonomy, possibly helping them to realize what are the deep interplays among vulnerabilities, identify which tools can detect them, what are the suggested mitigations and quickly obtain the most valuable bibliographic references to gather additional details. To demonstrate the feasibility of this use case, we implemented a first preliminary interactive visualization platform [71] where users can navigate our initial instantiation of the data model by traversing the relationships between vulnerabilities.[8]

Statistical Analysis. Our data model allows the development of several statistical analyses aimed at identifying interesting trends and patterns. For instance, the data model can support a study aimed at understanding how different vulnerabilities were exploited over time in different real-world attacks. Similarly, relationships between attacks, patches and mitigations can help uncover that mitigation approaches may vary depending on the nature of the attack, enabling a comparison between theoretical best practices and real-world mitigation strategies employed in response to actual threats. Moreover, by linking tools with vulnerabilities, the model may reveal how the community is reacting to known vulnerabilities with respect to tools development; this relationship could also help developers to realize what tools can possibly offer better vulnerability coverage. To showcase such analyses, we augmented our preliminary interactive visualization platform [71] with a time-based visualization of the references, tools, and attacks related to a vulnerability.

Taxonomy-driven Security Frameworks. Our data model is designed to help automatic tools exploit taxonomies. For instance, a taxonomy could drive the orchestration of different detection tools. Indeed, tools come with different trade-offs in terms of coverage, accuracy, and scalability, thus choosing the right mix of tools may be an essential strategic decision when having a limited computational budget during a security audit. One possible strategy could be:

(1) Given one instantiation of our data model, the orchestrator ranks the tools, choosing the ones with the highest number of linked vulnerabilities;
(2) The orchestrator executes the first tool from the ranking, aiming at maximizing the chances of finding a flaw;
(3) Whenever a vulnerability is detected, the orchestrator evaluates the associated *enabled_by* and *impacted_by* relationships, deciding whether it could be convenient to run one specific tool targeted on one *adjacent* vulnerability;
(4) The ranking is then updated, removing any tool executed during step (2) and step (3), and then step (2) is repeated until the time budget is exhausted.

Additionally, the orchestrator could raise an alarm whenever an *adjacent* vulnerability does not have any tool able to detect it, pinpointing a blind spot that may require manual investigation. Furthermore, tools' outputs could be refined to explicitly refer to the taxonomy, helping users understand the meaning of the results while also favouring comparison across tools' results. Finally, tools may suggest the most relevant mitigations for the detected flaws.

We implemented a proof-of-concept of our orchestration strategy [73] into SmartBugs [22], a framework integrating several detection tools. For instance, when considering the `SimpleDAO` contract shipped with SmartBugs, our preliminary taxonomy instantiation, our PoC ranks the tools, selecting Slither as the best candidate. After executing it, several vulnerabilities are detected, including *Low Level Calls*, which in turn could enable, e.g., *Mishandled Exception*. Hence, the orchestrator considers from the ranking the three tools able to detect *Mishandled Exception*, selecting sfuzz as the next tool for execution. This tool is indeed able to confirm the presence of such vulnerability. The orchestration then continues following the rest of our strategy. Additional details are available at [73].

4.2 Enabled Use Cases

Efficacy-based Tool Orchestration. The relationship between a tool and a vulnerability may be enhanced with an *efficacy* score, representing the efficacy of a tool in finding a vulnerability. Devising such scores requires likely to evaluate tools on large smart contract datasets, commonly recognized by the community as representative of the real-world DApps. Unfortunately, at this time, such a benchmark platform is not yet available. Nonetheless, we believe such scores may help to significantly improve the orchestration strategy proposed in the previous section.

Severity-based Prioritization. Traditional classification schemes often associate a severity score with each vulnerability. Consistently, our data model may devise an attribute representing the severity to

[8] The visualization platform assists users in analyzing the vast amount of heterogeneous data integrated within the data model.

support different prioritization processes. For instance, such score may help refine tool orchestration, aiming at focusing analysis time towards the most critical flaws. Moreover, scores from different vulnerabilities could be combined in presence of specific interplays suggested by our relationships. Finally, mitigation efforts could be strategically allocated taking into account the flaw severity. Unfortunately, there is not yet an established methodology for defining such severity scores in the blockchain ecosystem.

Overcome Taxonomies Syntactic Inconsistencies. In the case the community continues, at least in the short term, to define different and independent vulnerability taxonomies, we nonetheless believe there could be several advantages when such taxonomies decide to adopt our data model. Indeed, by performing a structural comparative analysis across the graphs representing the taxonomies, it should be likely possible to identify semantic equivalences of concepts despite inconsistent nomenclatures.

5 CONCLUSIONS

In this paper, we analyzed existing taxonomies on smart contract vulnerabilities. Our review revealed that such taxonomies generally lack coherence, with no standard nomenclature being commonly adopted. They lack a formal structure beyond basic descriptions, making them hard to integrate within automatic tools and hard to improve over time. Thus, we proposed a data model able to capture the key entities and associated relationships relevant to smart contract vulnerabilities. Our data model can support the construction of a well-structured taxonomy, opening the door to several relevant use cases. To demonstrate such value, we provided a preliminary instantiation, which can be updated by the community through a collaborative project [70]. Finally, we have implemented several PoCs to show the potential behind the presented use cases.

As future directions, we believe that our data model could be improved in several ways. First, we plan to revise it based on direct feedback from different blockchain security experts. A current limitation that we plan to fix is the support for better hierarchical views of the mitigations. Next, we may consider a transition from the current data model to a knowledge graph or to adopt an SQL-based schema, to enable specific queries and to ensure data integrity. Also, we would further work on integrating the taxonomy with automatic tools. Finally, the taxonomy would likely benefit from the definition of a severity scoring methodology, similar to what CWSS does for traditional software. This would enable to apply methods for vulnerability ranking, risk analysis and mitigation strategies recommendation by leveraging the taxonomy structure, which already facilitates a series of considerations regarding derived impacts and environmental factors.

ACKNOWLEDGMENTS

This work is partially supported by the Sapienza research project DYNASTY (protocol number RM123188F791C0F7).

REFERENCES

[1] Mouhamad Almakhour, Layth Sliman, Abed Ellatif Samhat, and Abdelhamid Mellouk. 2020. Verification of smart contracts: A survey. *Pervasive and Mobile Computing* 67 (2020), 101227. https://doi.org/10.1016/j.pmcj.2020.101227

[2] Mouhamad Almakhour, Layth Sliman, Abed Ellatif Samhat, and Abdelhamid Mellouk. 2023. A formal verification approach for composite smart contracts security using FSM. *Journal of King Saud University - Computer and Information Sciences* 35, 1 (2023), 70–86. https://doi.org/10.1016/j.jksuci.2022.08.029

[3] Nils Amiet. 2021. Blockchain Vulnerabilities in Practice. *Digital Threats: Research and Practice* 2, 2 (2021), 8:1–8:7. https://doi.org/10.1145/3407230

[4] Mauro C. Argañaraz, Mario Marcelo Berón, Maria João Varanda Pereira, and Pedro Rangel Henriques. 2020. Detection of Vulnerabilities in Smart Contracts Specifications in Ethereum Platforms. In *9th Symposium on Languages, Applications and Technologies (SLATE)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik. https://doi.org/10.4230/OASICS.SLATE.2020.2

[5] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. 2017. A Survey of Attacks on Ethereum Smart Contracts (SoK). In *Principles of Security and Trust (Lecture Notes in Computer Science)*. https://doi.org/10.1007/978-3-662-54455-6_8

[6] Priyanka Bose, Dipanjan Das, Yanju Chen, Yu Feng, Christopher Kruegel, and Giovanni Vigna. 2021. SAILFISH: Vetting Smart Contract State-Inconsistency Bugs in Seconds. *CoRR* abs/2104.08638 (2021). arXiv:2104.08638 https://arxiv.org/abs/2104.08638

[7] Lexi Brent, Neville Grech, Sifis Lagouvardos, Bernhard Scholz, and Yannis Smaragdakis. 2020. Ethainter: a smart contract security analyzer for composite vulnerabilities. In *Proc. of the 41st ACM SIGPLAN Conf. on Prog. Language Design and Implementation (PLDI 2020)*. https://doi.org/10.1145/3385412.3385990

[8] Lexi Brent, Anton Jurisevic, Michael Kong, Eric Liu, Francois Gauthier, Vincent Gramoli, Ralph Holz, and Bernhard Scholz. 2018. Vandal: A Scalable Security Analysis Framework for Smart Contracts. https://doi.org/10.48550/arXiv.1809.03981 arXiv:1809.03981 [cs]

[9] Vitalik Buterin. 2013. *Ethereum White Paper: A Next Generation Smart Contract & Decentralized Application Platform*. Technical Report. https://github.com/ethereum/wiki/wiki/White-Paper

[10] Stefanos Chaliasos, Marcos Antonios Charalambous, Liyi Zhou, Rafaila Galanopoulou, Arthur Gervais, Dimitris Mitropoulos, and Ben Livshits. 2023. Smart Contract and DeFi Security: Insights from Tool Evaluations and Practitioner Surveys. https://doi.org/10.48550/arXiv.2304.02981 arXiv:2304.02981 [cs]

[11] Huashan Chen, Marcus Pendleton, Laurent Njilla, and Shouhuai Xu. 2020. A Survey on Ethereum Systems Security: Vulnerabilities, Attacks, and Defenses. *Comput. Surveys* 53, 3 (2020), 67:1–67:43. https://doi.org/10.1145/3391195

[12] Jiachi Chen, Mingyuan Huang, Zewei Lin, Peilin Zheng, and Zibin Zheng. 2023. To Healthier Ethereum: A Comprehensive and Iterative Smart Contract Weakness Enumeration. http://arxiv.org/abs/2308.10227

[13] Jiachi Chen, Xin Xia, David Lo, John Grundy, Xiapu Luo, and Ting Chen. 2022. DefectChecker: Automated Smart Contract Defect Detection by Analyzing EVM Bytecode. *IEEE Transactions on Software Engineering* 48, 7 (2022), 2189–2207. https://doi.org/10.1109/TSE.2021.3054928

[14] Jiachi Chen, Xin Xia, David Lo, John Grundy, Xiapu Luo, and Ting Chen. 2022. Defining Smart Contract Defects on Ethereum. *IEEE Transactions on Software Engineering* 48, 1 (2022), 327–345. https://doi.org/10.1109/TSE.2020.2989002

[15] Ting Chen, Xiaoqi Li, Xiapu Luo, and Xiaosong Zhang. 2017. Under-optimized smart contracts devour your money. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)* (2017-02). 442–446. https://doi.org/10.1109/SANER.2017.7884650

[16] Hanting Chu, Pengcheng Zhang, Hai Dong, Yan Xiao, Shunhui Ji, and Wenrui Li. 2023. A survey on smart contract vulnerabilities: Data sources, detection and repair. *Information and Software Technology* 159 (2023), 107221. https://doi.org/10.1016/j.infsof.2023.107221

[17] Michael Coblenz. 2017. Obsidian: A Safer Blockchain Programming Language. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. 97–99. https://doi.org/10.1109/ICSE-C.2017.150

[18] Consensys. 2017. *Ethereum Smart Contract Best Practices*. https://consensys.github.io/smart-contract-best-practices/attacks/

[19] Consensys. 2020. Mythril. https://github.com/Consensys/mythril

[20] Crytic. 2024. Examples of Solidity security issues. https://github.com/crytic/not-so-smart-contracts

[21] Decurity. 2022. Semgrep rules for smart contracts based on DeFi exploits. https://github.com/Decurity/semgrep-smart-contracts

[22] Monika di Angelo, Thomas Durieux, João F. Ferreira, and Gernot Salzer. 2023. SmartBugs 2.0: An Execution Framework for Weakness Detection in Ethereum Smart Contracts. In *Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering (ASE 2023)*. https://doi.org/10.1109/ASE56229.2023.00060

[23] Monika di Angelo and Gernot Salzer. 2019. A Survey of Tools for Analyzing Ethereum Smart Contracts. In *2019 IEEE Int. Conf. on Decentralized Applications and Infrastructures (DAPPCON)*. https://doi.org/10.1109/DAPPCON.2019.00018

[24] Ardit Dika. 2017. *Ethereum Smart Contracts: Security Vulnerabilities and Security Tools*. Master thesis. https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2479191

[25] Wesley Dingman, Aviel Cohen, Nick Ferrara, Adam Lynch, Patrick Jasinski, Paul E. Black, and Lin Deng. 2019. Defects and Vulnerabilities in Smart Contracts, a Classification using the NIST Bugs Framework. *Int. Journal of Networked and Distributed Computing* (2019). https://doi.org/10.2991/ijndc.k.190710.003

[26] Thomas Durieux, João F. Ferreira, Rui Abreu, and Pedro Cruz. 2020. Empirical review of automated analysis tools on 47,587 Ethereum smart contracts. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE '20)*. https://doi.org/10.1145/3377811.3380364

[27] Ethereum. 2014. Solidity, the Smart Contract Programming Language. https://github.com/ethereum/solidity

[28] Josselin Feist, Gustavo Grieco, and Alex Groce. 2019. Slither: A Static Analysis Framework For Smart Contracts. In *2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*. https://doi.org/10.1109/WETSEB.2019.00008

[29] Manifold Finance. 2022. *manifoldfinance/defi-threat*. https://github.com/manifoldfinance/defi-threat original-date: 2020-08-25T19:09:04Z.

[30] Neville Grech, Lexi Brent, Bernhard Scholz, and Yannis Smaragdakis. 2019. Gigahorse: Thorough, Declarative Decompilation of Smart Contracts. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE. https://doi.org/10.1109/ICSE.2019.00120

[31] Neville Grech, Michael Kong, Anton Jurisevic, Lexi Brent, Bernhard Scholz, and Yannis Smaragdakis. 2018. MadMax: surviving out-of-gas conditions in Ethereum smart contracts. *Proceedings of the ACM on Programming Languages* 2 (2018), 116:1–116:27. Issue OOPSLA. https://doi.org/10.1145/3276486

[32] Ilya Grishchenko, Matteo Maffei, and Clara Schneidewind. 2018. Foundations and Tools for the Static Analysis of Ethereum Smart Contracts. In *Computer Aided Verification (Lect. Notes in Comp. Science)*. https://doi.org/10.1007/978-3-319-96145-3_4

[33] Ilya Grishchenko, Matteo Maffei, and Clara Schneidewind. 2018. A Semantic Framework for the Security Analysis of Ethereum smart contracts. Vol. 10804. 243–269. https://doi.org/10.1007/978-3-319-89722-6_10 arXiv:1802.08660 [cs]

[34] NCC Group. 2018. *DASP - TOP 10*. NCC Group. https://dasp.co/

[35] Bin Hu, Zongyang Zhang, Jianwei Liu, Yizhong Liu, Jiayuan Yin, Rongxing Lu, and Xiaodong Lin. 2020. A comprehensive survey on smart contract construction and execution: paradigms, tools, and systems. *Patterns* (2020). https://doi.org/10.1016/j.patter.2020.100179

[36] Tianyuan Hu, Bixin Li, Zhenyu Pan, and Chen Qian. 2023. Detect Defects of Solidity Smart Contract Based on the Knowledge Graph. *IEEE Transactions on Reliability* 73 (2023), 1–17. https://doi.org/10.1109/TR.2023.3233999

[37] Yongfeng Huang, Yiyang Bian, Renpu Li, J. Leon Zhao, and Peizhong Shi. 2019. Smart Contract Security: A Software Lifecycle Perspective. *IEEE Access* 7 (2019), 150184–150202. https://doi.org/10.1109/ACCESS.2019.2946988 Conference Name: IEEE Access.

[38] Runtime Verification Inc. 2018. *List of Security Vulnerabilities - GitHub*. https://github.com/runtimeverification/verified-smart-contracts/wiki/List-of-Security-Vulnerabilities

[39] Bo Jiang, Ye Liu, and W. K. Chan. 2018. ContractFuzzer: fuzzing smart contracts for vulnerability detection. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering* (New York, NY, USA) *(ASE '18)*. https://doi.org/10.1145/3238147.3238177

[40] kaden. 2019. *kadenzipfel/smart-contract-vulnerabilities*. https://github.com/kadenzipfel/smart-contract-vulnerabilities

[41] Mudabbir Kaleem, Anastasia Mavridou, and Aron Laszka. 2020. Vyper: A Security Comparison with Solidity Based on Common Vulnerabilities. In *2020 2nd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*. https://doi.org/10.1109/BRAINS49436.2020.9223278

[42] Sukrit Kalra, Seep Goel, Mohan Dhawan, and Subodh Sharma. 2018. ZEUS: Analyzing Safety of Smart Contracts. https://www.semanticscholar.org/paper/ZEUS%3A-Analyzing-Safety-of-Smart-Contracts-Kalra-Goel/f3f927adf4aac1146c9587fa646864a040c94fa6

[43] Johannes Krupp and Christian Rossow. 2018. teEther: Gnawing at Ethereum to Automatically Exploit Smart Contracts. https://www.usenix.org/conference/usenixsecurity18/presentation/krupp

[44] Satpal Singh Kushwaha, Sandeep Joshi, Dilbag Singh, Manjit Kaur, and Heung-No Lee. 2022. Systematic Review of Security Vulnerabilities in Ethereum Blockchain Smart Contract. *IEEE Access* 10 (2022), 6605–6621. https://doi.org/10.1109/ACCESS.2021.3140091

[45] Chao Liu, Han Liu, Zhao Cao, Zhong Chen, Bangdao Chen, and Bill Roscoe. 2018. ReGuard: Finding Reentrancy Bugs in Smart Contracts. In *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)* (2018-05). 65–68. https://ieeexplore.ieee.org/document/8449446 ISSN: 2574-1934.

[46] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. 2016. Making Smart Contracts Smarter. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA) *(CCS '16)*. Association for Computing Machinery, 254–269. https://doi.org/10.1145/

2976749.2978309
[47] Fuchen Ma, Meng Ren, Fu Ying, Wanting Sun, Houbing Song, Heyuan Shi, Yu Jiang, and Huizhong Li. 2023. V-Gas: Generating High Gas Consumption Inputs to Avoid Out-of-Gas Vulnerability. *ACM Trans. Internet Technol.* 23, 3, Article 40 (aug 2023), 22 pages. https://doi.org/10.1145/3511900
[48] MITRE. Accessed: 2024. *Common Vulnerabilities and Exposures (CVE).* https://cve.mitre.org/
[49] MITRE. Accessed: 2024. *Common Weakness Enumeration (CWE).* https://cwe.mitre.org/
[50] Mark Mossberg, Felipe Manzano, Eric Hennenfent, Alex Groce, Gustavo Grieco, Josselin Feist, Trent Brunson, and Artem Dinaburg. 2019. Manticore: A User-Friendly Symbolic Execution Framework for Binaries and Smart Contracts. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE).* https://doi.org/10.1109/ASE.2019.00133
[51] Dominik Muhs. 2023. *The Smart Contract Security Field Guide for Hackers - Smart Contract Security Field Guide.* https://scsfg.io/hackers/
[52] Sundas Munir and Walid Taha. 2023. Pre-deployment Analysis of Smart Contracts – A Survey. https://doi.org/10.48550/arXiv.2301.06079 arXiv:2301.06079 [cs]
[53] Satoshi Nakamoto. 2009. *Bitcoin: A Peer-to-Peer Electronic Cash System.* Technical Report. http://www.bitcoin.org/bitcoin.pdf
[54] Chaals Nevile. 2024. *EEA EthTrust Security Levels Specification v-after-2 Editor's Draft.* https://entethalliance.github.io/eta-registry/security-levels-spec.html
[55] Tai D. Nguyen, Long H. Pham, Jun Sun, Yun Lin, and Quang Tran Minh. 2020. sFuzz: an efficient adaptive fuzzer for solidity smart contracts. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE '20).* 778–788. https://doi.org/10.1145/3377811.3380334
[56] NIST. 2021. *The Bugs Framework.* https://www.nist.gov/itl/ssd/software-quality-group/samate/bugs-framework
[57] NVD. Accessed: 2024. *National Vulnerability Database (NVD).* https://www.nist.gov/programs-projects/national-vulnerability-database-nvd
[58] National Institute of Standards and Technology (NIST). Accessed: 2024. *Software Vulnerability.* https://csrc.nist.gov/glossary/term/software_vulnerability
[59] National Institute of Standards and Technology (NIST). Accessed: 2024. *Software Weakness.* https://csrc.nist.gov/glossary/term/weakness
[60] OpenZeppelin. 2018. *Math - OpenZeppelin Docs.* https://docs.openzeppelin.com/contracts/2.x/api/math
[61] OpenZeppelin. 2024. *OpenZeppelin/openzeppelin-contracts.* https://github.com/OpenZeppelin/openzeppelin-contracts original-date: 2016-08-01T20:54:54Z.
[62] palkeo. 2018. *palkeo/pakala.* https://github.com/palkeo/pakala original-date: 2018-12-03T17:31:46Z.
[63] Daniel Perez and Benjamin Livshits. 2021. Smart Contract Vulnerabilities: Vulnerable Does Not Imply Exploited. In *30th USENIX Security Symposium (USENIX Security 21).*
[64] Purathani Praitheeshan, Lei Pan, Jiangshan Yu, Joseph Liu, and Robin Doss. 2020. Security Analysis Methods on Ethereum Smart Contract Vulnerabilities: A Survey. https://doi.org/10.48550/arXiv.1908.08605 arXiv:1908.08605 [cs]
[65] Sigma Prime. 2018. *sigp/solidity-security-blog.* https://github.com/sigp/solidity-security-blog
[66] Protofire. 2024. *protofire/solhint.* https://github.com/protofire/solhint original-date: 2017-10-16T11:48:44Z.
[67] Peng Qian, Zhenguang Liu, Qinming He, Butian Huang, Duanzheng Tian, and Xun Wang. 2022. Smart Contract Vulnerability Detection Technique: A Survey. https://doi.org/10.13328/j.cnki.jos.006375 arXiv:2209.05872 [cs]
[68] Heidelinde Rameder. 2021. *Systematic review of ethereum smart contract security vulnerabilities, analysis methods and tools.* Thesis. https://doi.org/10.34726/hss.2021.86784
[69] Heidelinde Rameder, Monika di Angelo, and Gernot Salzer. 2022. Review of Automated Vulnerability Analysis of Smart Contracts on Ethereum. *Frontiers in Blockchain* 5 (2022). https://www.frontiersin.org/articles/10.3389/fbloc.2022.814977
[70] Claudia Ruggiero, Pietro Mazzini, Emilio Coppa, Simone Lenti, and Silvia Bonomi. 2024. *Smart Vulnerabilities.* https://ds-square.github.io/smart-vulnerabilities/
[71] Claudia Ruggiero, Pietro Mazzini, Emilio Coppa, Simone Lenti, and Silvia Bonomi. 2024. *Smart Vulnerabilities Interactive Viewer.* https://ds-square.github.io/smart-vulnerabilities/main
[72] Claudia Ruggiero, Pietro Mazzini, Emilio Coppa, Simone Lenti, and Silvia Bonomi. 2024. *Smart Vulnerabilities JSON Schema.* https://github.com/ds-square/smart-vulnerabilities/blob/main/schema.json
[73] Claudia Ruggiero, Pietro Mazzini, Emilio Coppa, Simone Lenti, and Silvia Bonomi. 2024. *Smartbugs Orchestrator.* https://github.com/ds-square/smartbugs-orchestrator
[74] Claudia Ruggiero, Pietro Mazzini, Emilio Coppa, Simone Lenti, and Silvia Bonomi. 2024. Technical Report. https://github.com/ds-square/smart-vulnerabilities/blob/main/SoK__A_Unified_Data_Model_for_Smart_Contract_Vulnerability_Taxonomies.pdf
[75] Sarwar Sayeed, Hector Marco-Gisbert, and Tom Caira. 2020. Smart Contract: Attacks and Protections. *IEEE Access* 8 (2020), 24416–24427. https://doi.org/10.1109/ACCESS.2020.2970495 Conference Name: IEEE Access.
[76] Clara Schneidewind, Ilya Grishchenko, Markus Scherer, and Matteo Maffei. 2020. eThor: Practical and Provably Sound Static Analysis of Ethereum Smart Contracts. https://doi.org/10.48550/arXiv.2005.06227 arXiv:2005.06227 [cs]
[77] SECBIT Labs. 2018. *A Collection of Risks and Vulnerabilities in ERC20 Token Contracts.* https://github.com/sec-bit/awesome-buggy-erc20-tokens/blob/master/ERC20_token_issue_list.md
[78] Rajeev Secureum. 2021. *Security Pitfalls & Best Practices 101.* https://secureum.substack.com/p/security-pitfalls-and-best-practices-101
[79] Rajeev Secureum. 2021. *Security Pitfalls & Best Practices 201.* https://secureum.substack.com/p/security-pitfalls-and-best-practices-201
[80] Ilya Sergey, Amrit Kumar, and Aquinas Hobor. 2018. Scilla: a Smart Contract Intermediate-Level LAnguage. (2018). https://doi.org/10.48550/arXiv.1801.00687 arXiv:1801.00687 [cs]
[81] SmartContractSecurity. 2019. *Smart Contract Weakness Classification (SWC).* https://swcregistry.io/
[82] SmartDec. 2018. *smartdec/classification.* https://github.com/smartdec/classification
[83] ETH Zurich SRI Lab. 2020. *eth-sri/securify2.* https://github.com/eth-sri/securify2
[84] Mirko Staderini, Caterina Palli, and Andrea Bondavalli. 2020. Classification of Ethereum Vulnerabilities and their Propagations. In *2020 Second International Conference on Blockchain Computing and Applications (BCCA).* https://doi.org/10.1109/BCCA50787.2020.9274458
[85] Mirko Staderini, Andràs Pataricza, and Andrea Bondavalli. 2022. Security Evaluation and Improvement of Solidity Smart Contracts. https://doi.org/10.2139/ssrn.4038087
[86] Tianyu Sun and Wensheng Yu. 2020. A Formal Verification Framework for Security Issues of Blockchain Smart Contracts. *Electronics* 9, 2 (2020), 255. https://doi.org/10.3390/electronics9020255 Number: 2 Publisher: Multidisciplinary Digital Publishing Institute.
[87] Xiangyan Tang, Ke Zhou, Jieren Cheng, Hui Li, and Yuming Yuan. 2021. The Vulnerabilities in Smart Contracts: A Survey. In *Advances in Artificial Intelligence and Security*, Xingming Sun, Xiaorui Zhang, Zhihua Xia, and Elisa Bertino (Eds.). https://doi.org/10.1007/978-3-030-78621-2_14
[88] Sergei Tikhomirov, Ekaterina Voskresenskaya, Ivan Ivanitskiy, Ramil Takhaviev, Evgeny Marchenko, and Yaroslav Alexandrov. 2018. SmartCheck: static analysis of ethereum smart contracts. In *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain.* 9–16. https://doi.org/10.1145/3194113.3194115
[89] Christof Ferreira Torres, Antonio Ken Iannillo, Arthur Gervais, and Radu State. 2021. ConFuzzius: A Data Dependency-Aware Hybrid Fuzzer for Smart Contracts. https://doi.org/10.1109/EuroSP51992.2021.00018
[90] Christof Ferreira Torres, Julian Schütte, and Radu State. 2018. Osiris: Hunting for Integer Bugs in Ethereum Smart Contracts. In *Proceedings of the 34th Annual Computer Security Applications Conference.* https://doi.org/10.1145/3274694.3274737
[91] Petar Tsankov, Andrei Dan, Dana Drachsler-Cohen, Arthur Gervais, Florian Bünzli, and Martin Vechev. 2018. Securify: Practical Security Analysis of Smart Contracts. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security.* 67–82. https://doi.org/10.1145/3243734.3243780
[92] Nuno Veloso. 2021. Conkas: A Modular and Static Analysis Tool for Ethereum Bytecode. https://fenix.tecnico.ulisboa.pt/downloadFile/1689244997262417/94080-Nuno-Veloso_resumo.pdf
[93] Fernando Richter Vidal, Naghmeh Ivaki, and Nuno Laranjeiro. 2023. OpenSCV: An Open Hierarchical Taxonomy for Smart Contract Vulnerabilities. arXiv:2303.14523 [cs] http://arxiv.org/abs/2303.14523
[94] Ziwei Wang, Haotian Lu, and Xuetao Wei. 2023. Ethereum DeFi Apps in the Wild: Profiling and Implications. In *2023 IEEE International Conference on Metaverse Computing, Networking and Applications (MetaCom).* 388–392. https://doi.org/10.1109/MetaCom57706.2023.00074
[95] Wikipedia. 2024. *Jaccard index.* https://en.wikipedia.org/w/index.php?title=Jaccard_index&oldid=1196092673 Page Version ID: 1196092673.
[96] Zhuo Zhang, Brian Zhang, Wen Xu, and Zhiqiang Lin. 2023. Demystifying Exploitable Bugs in Smart Contracts. 615–627. https://doi.org/10.1109/ICSE48619.2023.00061
[97] Haozhe Zhou, Amin Milani Fard, and Adetokunbo Makanju. 2022. The State of Ethereum Smart Contracts Security: Vulnerabilities, Countermeasures, and Tool Support. *Journal of Cybersecurity and Privacy* 2, 2 (2022), 358–378. https://doi.org/10.3390/jcp2020019