SCRUBD: Smart Contracts Reentrancy and Unhandled Exceptions Vulnerability Dataset

Chavhan Sujeet Yashavant[1], MitrajSinh Chavda[1], Saurabh Kumar[2], Amey Karkare[1], Angshuman Karmakar[1]
[1]*Department of Computer Science and Engineering*, *Indian Institute of Technology Kanpur*, Kanpur, India
Email: {sujeetc, mbchavda, karkare, angshuman}@cse.iitk.ac.in
[2]*Department of Computer Science and Engineering*, *Indian Institute of Technology Hyderabad*, Hyderabad, India
Email: saurabhkr@cse.iith.ac.in

***Abstract*—Smart Contracts (SCs) handle transactions in the Ethereum blockchain worth millions of United States dollars, making them a lucrative target for attackers seeking to exploit vulnerabilities and steal funds. The Ethereum community has developed a rich set of tools to detect vulnerabilities in SCs, including reentrancy (RE) and unhandled exceptions (UX). A dataset of SCs labeled with vulnerabilities is needed to evaluate the tools' efficacy. Existing SC datasets with labeled vulnerabilities have limitations, such as covering only a limited range of vulnerability scenarios and containing incorrect labels. As a result, there is a lack of a standardized dataset to compare the performances of these tools. Our dataset, SCRUBD, aims to fill this gap. SCRUBD is a dataset of real-world SCs and synthesized SCs labeled with RE and UX vulnerabilities. The real-world SC dataset is labeled through crowdsourcing, followed by manual inspection by an experienced SC programmer, and covers both RE and UX vulnerabilities. On the other hand, the synthesized dataset is carefully crafted to cover various RE scenarios only. Using SCRUBD, we compared the performance of six popular vulnerability detection tools. Based on our study, we found that Slither outperforms other tools on a crowdsourced dataset in detecting RE vulnerabilities, while Sailfish outperforms other tools on a manually synthesized dataset for detecting RE. For UX vulnerabilities, Slither outperforms all other tools.**

***Index Terms*—Smart Contract Vulnerabilities, Reentrancy, Unhandled Exceptions, Dataset**

I. Introduction

Public blockchains like Ethereum host millions of smart contracts (SCs) that can be invoked by anyone via public functions [1]. We focus on Ethereum because it is the leading platform for SCs, which has achieved a market capitalization of more than 350 billion United States Dollars[1]. SCs manage Ether, Ethereum's cryptocurrency, which is valued in various fiat currencies like USD (United States Dollars), EUR (Euro), and INR (Indian Rupees). Hence, SCs are crucial for Ethereum. However, their significance makes them lucrative targets for malicious actors seeking financial gain or intending to cause disruption. Consequently, attacks on Ethereum SCs that target vulnerabilities in SCs have sharply increased over recent years [2]–[5]. This increase in attacks highlights the urgent need for stronger security measures and robust protocols in the Ethereum ecosystem.

To strengthen the security of SCs, the blockchain community has developed a diverse array of tools to detect vulnerabilities in SCs [6]–[12]. Testing using a quality test suite (dataset) is integral to any tool development process. The developers use these datasets to assess efficacy of vulnerability detection tools. However, it is difficult to compare these metrics for different tools as each tool relies on a different dataset [6], [9], [12], [13]. For a fair and consistent comparison of vulnerability detection tools, there is a pressing need to create a high-quality sound dataset. Further, a new vulnerability detection tool can be tested on this dataset and refined progressively.

TABLE I: Total Vulnerable (V) and Non-Vulnerable (NV) Functions in SCRUBD/CD and SCRUBD/SD

Dataset	Methodology	RE	UX
SCRUBD/CD	Crowdsourcing	245 V, 501 NV	275 V, 291 NV
SCRUBD/SD	Synthesized Dataset	155 V, 84 NV	N/A

We outline the limitations of some well-known SC vulnerability datasets in Section II. To overcome the limitations of existing SC vulnerability datasets, we present SCRUBD. In this work, we focus on RE and UX vulnerabilities, as an earlier study [14] has shown that they are the most commonly exploited vulnerabilities. Our contributions are as follows:

1) A dataset of SCs labeled with Reentrancy and Unhandled Exceptions vulnerabilities, called SCRUBD. It consists of two sub-parts: SCRUBD/CD (crowdsourced dataset) and SCRUBD/SD (synthesized dataset). SCRUBD/CD covers RE and UX vulnerabilities, whereas SCRUBD/SD covers only RE vulnerabilities. Table I shows the total number of vulnerable and non-vulnerable functions in each of the above datasets.
2) A comparison of the performance of six SC vulnerability analysis tools using SCRUBD. In our experiments to detect RE vulnerabilities, Slither outperforms tools for SCRUBD/CD, while Sailfish excels for SCRUBD/SD. Slither performs best in detecting UX vulnerabilities that are present only in SCRUBD/CD (see Section V). We identified misleading labels in the dataset [15], a finding that has been confirmed by the authors (see Section II). Our dataset and website are now publicly accessible, along with limitations of current datasets[2].

[1]https://coinmarketcap.com/currencies/ethereum

[2]https://github.com/sujeetc/SCRUBD, https://scaudit.cse.iitk.ac.in

II. Limitations of Existing RE and UX Datasets

This section outlines the limitations of existing datasets.

A. *Turn-The-Rudder Dataset*

Zheng et al. [15] used crowdsourcing to annotate smart contracts (SCs) for RE. After crowdsourcing, they discovered that 41 contracts have RE out of 21,212, with each contract having a vulnerable function. However, when we manually reviewed the non-vulnerable SCs, we found that they contained the exact copy of a few functions from the vulnerable contracts. We conveyed our findings to authors with five such entries, and they confirmed our findings and updated their GitHub [16]. So there are *46* vulnerable functions now. We applied function-level de-duplication and identified 20 unique functions with reentrancy. Upon semantically matching these 20 functions, we found that only 14 unique functions remained. We found that the non-vulnerable section contains 71 functions that match exactly with vulnerable functions flagged by the dataset and therefore are potentially vulnerable.

B. *Manually Injected (MI) Dataset*

Ren et al. [13] created two labelled datasets. One dataset is called Manually Injected (MI) bugs, created by injecting bugs into different SC locations. MI dataset has seven different types of vulnerabilities with 50 contracts per vulnerability. The paper mentions that the logic for vulnerabilities is simple and involves only a few obvious patterns. Hence, this dataset does not cover all RE and UX scenarios. Another issue with the MI dataset is the absence of non-vulnerable functions, which makes it impossible to check the false positives of existing tools. The second dataset (CVE), does not include RE or UX vulnerabilities and, therefore, is not considered.

C. *Majority Voting Dataset*

Ren et al. [17] employed tools Oyente [7], Mythril [10], Securify [8], Smartcheck [9], Pied-Piper [18] to detect vulnerability in SCs. Among these tools, only three support RE and UX vulnerabilities [19], [20]. A contract is classified as vulnerable if majority of the tools concur on presence of a vulnerability. However, the reliability of this approach is compromised by the high rate of false positives, as it can lead to erroneous classifications of contracts as vulnerable, thereby reducing the framework's effectiveness in accurately identifying genuine vulnerabilities.

D. *Smartbugs Dataset*

Durieux et al. [20] crafted an annotated and non-annotated SCs dataset. The annotated part contains 31 contracts tagged with RE vulnerability. Total functions in these contracts are 177, out of which only 31 are vulnerable for RE. Moreover, we found a function named `WithdrawToHolder(address,uint)` in the dataset that is marked as vulnerable for RE even though the function is non-vulnerable [21]. It is protected by the modifier that restricts function calling i.e. `onlyOwner` [22]. A similar concept is demonstrated in Listing 2, in the function `non_vuln_onlyOwner()`.

III. SCRUBD/CD: SCs Labelled Using Crowdsourcing

To collect SCs for SCRUBD/CD, we developed a website that lists SC functions and asked users to flag them as vulnerable or non-vulnerable. The website lists a function in a SC only if it is flagged as vulnerable by at least one of the following tools: Conkas [23], Mythril [10], Sailfish [12], Slither [6], Solhint [24]. This was assigned as a course assignment in a graduate-level course. An experienced researcher manually reviewed and validated the students' answers. The resulting crowdsource dataset includes 469 contracts. Within 469 contracts, 746 functions are labelled for RE vulnerability and 566 functions are labelled for UX vulnerability (see Table I).

A. *Data Source*

We now explain the collection methodology for contracts in SCRUBD/CD.

```
SELECT contracts.address, COUNT(1) AS tx_count
FROM `bigquery-public-data.crypto_ethereum.contracts
    ` AS contracts
JOIN `bigquery-public-data.crypto_ethereum.
    transactions` AS transactions
ON (transactions.to_address = contracts.address)
GROUP BY contracts.address HAVING tx_count > 1 ORDER
     BY tx_count DESC
```

Listing 1: Query to Retrieve SC Addresses with Transaction Counter Greater Than One from Google BigQuery

We utilized the *crypto_ethereum* table from Google BigQuery [25], a leading platform that provides access to diverse organizational databases. The *crypto_ethereum* table from Google BigQuery serves as a comprehensive repository for the most up-to-date Ethereum SC addresses. To filter out dummy SCs we consider SCs with transaction count greater than one. Running the query in Listing 1 on Google BigQuery yielded 43,83,982 SC addresses.

We used the Etherscan API[3] to retrieve the contract source code, extracting 21,58,515 source codes out of 43,83,982 contracts, as many SCs do not make their source code public and due to Etherscan API extraction limitations. We performed de-duplication, following the approach adopted from [20], to obtain unique smart contracts (SCs). This process involved removing newlines, spaces, and comments from the SCs, followed by exactly matching the cleaned versions to identify and eliminate duplicates. After performing de-duplication on extracted SCs, we got 4,54,895 unique SCs. We labelled 469 contracts, chosen for their mix of simple to complex functions and coverage across different Solidity versions. Our rigorous procedure limited the number we could annotate. We plan to continue the labeling process to improve the dataset.

B. *Tools Used*

Following previous work [20], we select tools based on the criteria that they are publicly released with a command-line

[3]https://docs.etherscan.io/api-endpoints/contracts#get-contract-source-code-for-verified-contract-source-codes

interface (CLI), accept Solidity[4] SCs as input, and are scalable, actively maintained, with clear setup instructions.

We selected Slither [6], Mythril [10], Solhint [24], Conkas [23], and Sailfish [12] for our analysis, while excluding dynamic tools such as ityfuzz [26], Reguard [27], SoliAudit [28], ConFuzzius [29], and ContractFuzzer [30] due to their scalability limitations. We excluded Oyente [7] and Securify [8] as they are no longer maintained.

C. SCRUBD/CD *Methodology*

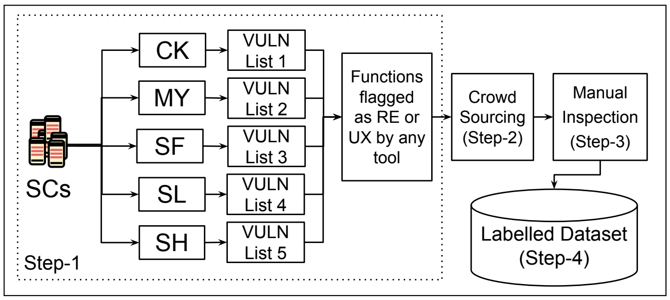


Fig. 1: Methodology to create SCRUBD/CD (Conkas: CK, Mythril: MY, Sailfish: SF, Slither: SL, Solhint: SH)

Figure 1 illustrates our rigorous methodology for creating SCRUBD/CD. The goal of employing the crowdsourcing approach is to gather input from graduate-level Computer Science and Engineering (CSE) students to label real-world SCs. The steps are as follows:

1) We initially taught students the basics of blockchain, smart contracts, and vulnerabilities in a graduate-level course in the Department of CSE (Step-0).
2) The SC vulnerability task was assigned via a website, where functions flagged as RE or UX by at least one tool (Solhint, Sailfish, Mythril, Conkas, Slither) were reviewed by students. Each student analyzed 20 functions, with a maximum of three students per function (Step-1).
3) We gathered input from 63 students (Step-2). The annotations were reviewed and validated by one of the authors, an experienced SC programmer (Step-3).
4) The conflicting labels were finalized after one-to-one meeting with students. As a result, we have dataset with high confidence. Moreover, we have proper comments explaining why a vulnerability exists or does not exist for non-trivial functions (Step-4).

Understanding a SC function is an intense task and our crowdsourcing methodology is quite rigorous. Therefore we cannot expect humans to be consistent if a large number of SC functions are given for annotation. Hence, we decided to give *20* functions to each student. The total number of functions flagged as vulnerable for RE or UX by at least one tool is 2080 from 469 contracts. Due to the reasons specified above, we were able to annotate 746 functions for RE and 566 for UX from 469 contracts. Out of 746 functions, 245 are vulnerable to RE while the rest 501 are non-vulnerable. Out of the 746 functions for RE, 566 were annotated with the help of students, while 180 were annotated directly by an experienced SC programmer. Within the same 469 contracts, we have 566 functions labelled for UX Vulnerability. Out of 566 functions, 275 are vulnerable to UX, whereas the rest 291 are non-vulnerable. All UX functions were annotated with the help of students (see Table I).

[4]Solidity is a widely used programming language for writing SCs.

IV. SCRUBD/SD: Manually Synthesized Reentrancy Dataset

To create SCRUBD/SD, we analyzed various scenarios that lead to RE vulnerabilities and created manually synthesized test cases for both vulnerable and non-vulnerable RE scenarios. Thus, SCRUBD/SD covers corner cases of RE vulnerability which are not present in SCRUBD/CD. In particular, it includes cases where a minor modification can convert a test case from vulnerable to non-vulnerable (and vice versa). SCRUBD/SD contains a total of 239 cases, out of which 155 are vulnerable and 84 are non-vulnerable (Table I).

A. *Methodology*

Two of the authors are experienced SC programmers and have studied RE vulnerability in-depth. Using their experience, they manually created various scenarios that result in RE and other scenarios that are very similar but do not result in RE. We have compiled these scenarios into a dataset focused on RE vulnerabilities, which covers diverse cases, such as data dependencies, control dependencies, RE vulnerabilities through require/assert statements, and multi-call issues. We believe this dataset encompasses nearly all potential RE scenarios. SCRUBD/SD contains a total of 239 functions, out of which 155 are vulnerable and 84 are non-vulnerable. Table I describes the result of annotations. SCRUBD/SD can be used as a test suite to check if the existing tools cover complex as well as simple RE scenarios. SCRUBD/SD currently does not handle UX.

```
modifier onlyOwner() {
  require(msg.sender == owner, "Only the owner can execute this.");
  _; }
function vuln_data_dep() public {  // parameter of call is data detependent on st updated post call
  msg.sender.call{value: state_var}("");
  state_var = state_var - 1;  }
function vuln_control_dep() public {
  if (state_var_1 > 0)
    msg.sender.call{value: state_var_2}("");
  state_var_1 = state_var_1 - 1; } // State variable is updated after the call
function non_vuln_onlyOwner() public onlyOwner {
  msg.sender.call{value: state_var}("");
  state_var = state_var - 1; }
function vuln_multi_call(address address_1, address address_2) public {
  address_1.call{value:address(this).balance/2}("");
  // Address_1 can repeatedly reenter until all Ether is drained.
  address_2.call{value:address(this).balance/2}("");
}
```

Listing 2: Example RE Scenarios in SCRUBD/SD

B. Reentrancy Scenarios

1) RE Induced by Data Dependency: The function `vuln_data_dep()` in Listing 2 illustrates a data dependency scenario, where the external call's parameter (`state_var`) is updated after the call. If an attacker reenters before the update, they can receive Ether they should not get. Thus, *the external call's parameter depends on the state variable updated after the call.*

2) RE Induced by Control Dependency: The function `vuln_control_dep()` from Listing 2 shows a function in which *an external call is control dependent on state variable updated after the external call (`state_var_1`).* Here, an attacker can reenter the function before the updation of an external call, and the condition will always be true. Meanwhile, the condition can become false in the non-reentrant scenario after updating the state variable.

3) RE Induced by Modifier that Restricts Function Calling: The function `non_vuln_onlyOwner()` from Listing 2 shows an example where the function is protected by `onlyOwner()` modifier [22]. Here, the state variable `owner` denotes the address of the owner of the contract. The modifier ensures that only the owner can call the function. An attacker cannot reenter such a function because of protection from the modifier.

4) RE Induced by Multi Call: The function `vuln_multi_call()` from Listing 2 shows an example of reentrancy vulnerability due to multiple external calls that belong to different addresses. In the case of sequential execution `address_1` will get half ether and `address_2` will get half ether. If address `address_1` is malicious then it can reenter the function till it drains all Ether from the contract. Address `address_2` won't get any Ether.

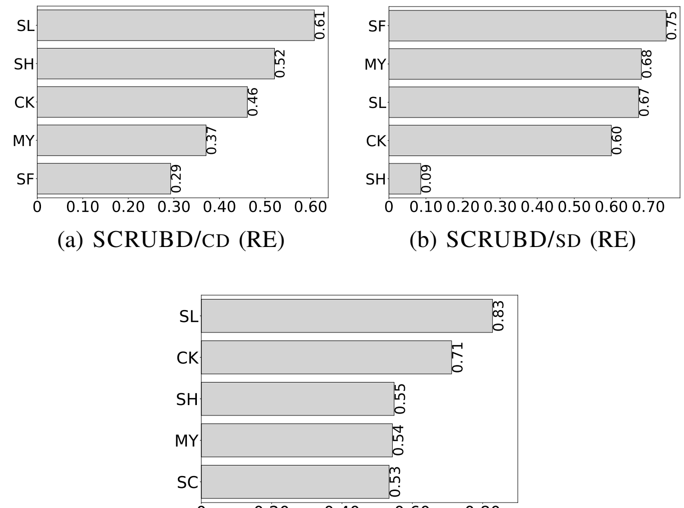
A detailed technical report [31] presents more scenarios.

V. Evaluation of Existing Tools using SCRUBD

This section shows F1-Score of existing tools (Slither [6], Mythril [10], Sailfish [12], Conkas [23], Smartcheck [9], Solhint [24]) while detecting RE and UX vulnerabilities from SCRUBD.

Figure 2a shows the F1-Score of existing tools while detecting RE on SCRUBD/CD. The F1-Score is a metric used to evaluate the performance of a classification model, and it is the harmonic mean of precision and recall [32]. Slither (SL) outperforms other tools because it over-approximates RE. SCRUBD/CD do not cover all corner cases of RE. Figure 2b shows the F1-Score of existing tools on SCRUBD/SD. Here, Sailfish outperforms other tools because it handles corner cases of RE present in SCRUBD/SD. Smartcheck [9] does not support detection of RE and hence omitted.

Figure 2c shows the performance of existing tools on UX vulnerabilities from SCRUBD/CD. Sailfish does not support the detection of UX and hence omitted. Here, we can see that Slither outperforms other tools in detecting UX vulnerabilities. Sailfish [12] does not support detection of UX and hence omitted.



(a) SCRUBD/CD (RE) (b) SCRUBD/SD (RE)

(c) SCRUBD/CD (UX)

Fig. 2: F1-Scores of Slither(SL), Sailfish(SF), Solhint(SH), Mythril(MY), Conkas(CK), Smartcheck(SC) using SCRUBD

VI. Applications of SCRUBD

This section discusses applications of SCRUBD.

1) **Evaluation of New and Existing Tools:** Researchers can check if existing as well as new tools cover the complex scenarios of Reentrancy Vulnerability and Unhandled Exceptions vulnerability using SCRUBD.
2) **Training of Machine Learning based Tools:** Machine learning-based SC vulnerability analysis tools [33]–[35] require a labelled dataset to be trained. SCRUBD can serve as a foundation for researchers to learn or train machine learning-based tools.

VII. Conclusion and Future Work

We introduced SCRUBD, a dataset designed to capture a wide range of RE and UX vulnerabilities in SCs. SCRUBD consists of two parts: SCRUBD/CD and SCRUBD/SD. The SCRUBD/CD portion contains real-world SCs labelled for RE and UX vulnerabilities, created using crowdsourcing, while SCRUBD/SD was developed by carefully crafting additional RE scenarios not covered in SCRUBD/CD. SCRUBD includes 985 functions labelled with RE vulnerability and 566 functions labelled with UX vulnerability. We found that Slither performs best at detecting RE and UX in contracts from SCRUBD/CD, while Sailfish excels in detecting RE from SCRUBD/SD. Going forward, we plan to conduct a more detailed empirical evaluation of these tools using SCRUBD.

VIII. Acknowledgement

We are grateful for valuable feedback from all anonymous reviewers. We thank students in 2023-24 offerings of courses CS731, CS628, and CS989 at IIT Kanpur for their labelling efforts. We extend special thanks to Rohit Singh, Vishal Gupta, and Arun KP for their insightful reviews of our work.

REFERENCES

[1] A. Miller, Z. Cai, and S. Jha, "Smart contracts and opportunities for formal methods," in *Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice*, T. Margaria and B. Steffen, Eds. Cham: Springer International Publishing, 2018, pp. 280–299.

[2] H. Chen, M. Pendleton, L. Njilla, and S. Xu, "A survey on ethereum systems security: Vulnerabilities, attacks, and defenses," *ACM Comput. Surv.*, vol. 53, no. 3, jun 2020.

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

[4] H. Yuan, "Two major attacks on the ethereum smart contract application layer," in *Third International Conference on Machine Learning and Computer Application (ICMLCA 2022)*, vol. 12636. SPIE, 2023, pp. 1230–1237.

[5] T. Jiao, Z. Xu, M. Qi, S. Wen, Y. Xiang, and G. Nan, "A survey of ethereum smart contract security: Attacks and detection," *Distrib. Ledger Technol.*, vol. 3, no. 3, Sep. 2024.

[6] J. Feist, G. Greico, and A. Groce, "Slither: A static analysis framework for smart contracts," in *Proceedings of the 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain*, ser. WETSEB '19. IEEE Press, 2019, p. 8–15.

[7] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 254–269.

[8] 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. New York, NY, USA: Association for Computing Machinery, 2018, p. 67–82.

[9] S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, R. Takhaviev, E. Marchenko, and Y. Alexandrov, "Smartcheck: Static analysis of ethereum smart contracts," in *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain*, ser. WETSEB '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 9–16.

[10] ConsenSys, "Mythril," 2022. [Online]. Available: https://github.com/ConsenSys/mythril-classic

[11] I. Nikolić, A. Kolluri, I. Sergey, P. Saxena, and A. Hobor, "Finding the greedy, prodigal, and suicidal contracts at scale," in *Proceedings of the 34th Annual Computer Security Applications Conference*, ser. ACSAC '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 653–663.

[12] P. Bose, D. Das, Y. Chen, Y. Feng, C. Kruegel, and G. Vigna, "Sailfish: Vetting smart contract state-inconsistency bugs in seconds," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 161–178.

[13] M. Ren, Z. Yin, F. Ma, Z. Xu, Y. Jiang, C. Sun, H. Li, and Y. Cai, "Empirical evaluation of smart contract testing: What is the best choice?" in *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 566–579.

[14] D. Perez and B. Livshits, "Smart contract vulnerabilities: Vulnerable does not imply exploited," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 1325–1341.

[15] Z. Zheng, N. Zhang, J. Su, Z. Zhong, M. Ye, and J. Chen, "Turn the rudder: A beacon of reentrancy detection for smart contracts on ethereum," in *Proceedings of the 45th International Conference on Software Engineering*, ser. ICSE '23. IEEE Press, 2023, p. 295–306.

[16] ——, https://github.com/InPlusLab/ReentrancyStudy-Data/commit/5b6be0f6be24591f1455e3467d956c397357abd3, 2023.

[17] M. Ren, F. Ma, Z. Yin, Y. Fu, H. Li, W. Chang, and Y. Jiang, "Making smart contract development more secure and easier," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 1360–1370.

[18] F. Ma, M. Ren, L. Ouyang, Y. Chen, J. Zhu, T. Chen, Y. Zheng, X. Dai, Y. Jiang, and J. Sun, "Pied-piper: Revealing the backdoor threats in ethereum erc token contracts," *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 3, Apr. 2023.

[19] R. M. Parizi, A. Dehghantanha, K.-K. R. Choo, and A. Singh, "Empirical vulnerability analysis of automated smart contracts security testing on blockchains," in *Proceedings of the 28th Annual International Conference on Computer Science and Software Engineering*, ser. CASCON '18. USA: IBM Corp., 2018, p. 103–113.

[20] T. Durieux, J. a. F. Ferreira, R. Abreu, and P. Cruz, "Empirical review of automated analysis tools on 47,587 ethereum smart contracts," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ser. ICSE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 530–541.

[21] ——, https://github.com/smartbugs/smartbugs-curated/blob/main/dataset/reentrancy/0x627fa62ccbb1c1b04ffaecd72a53e37fc0e17839.sol, 2020.

[22] R. I. (https://www.risein.com/), "onlyowner-modifier," 2024. [Online]. Available: https://www.risein.com/courses/solidity-fundamentals/onlyOwner-modifier

[23] N. Veloso, "Conkas," 2024. [Online]. Available: https://github.com/nveloso/conkas

[24] ProtoFire, "Solhint," 2024. [Online]. Available: https://github.com/protofire/solhint

[25] G. LLC, "Bigquery enterprise data warehouse — google cloud," 2024. [Online]. Available: https://cloud.google.com/bigquery

[26] C. Shou, S. Tan, and K. Sen, "Ityfuzz: Snapshot-based fuzzer for smart contract," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2023, pp. 322–333.

[27] C. Liu, H. Liu, Z. Cao, Z. Chen, B. Chen, and B. Roscoe, "Reguard: finding reentrancy bugs in smart contracts," in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, ser. ICSE '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 65–68.

[28] J.-W. Liao, T.-T. Tsai, C.-K. He, and C.-W. Tien, "Soliaudit: Smart contract vulnerability assessment based on machine learning and fuzz testing," in *2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*, 2019, pp. 458–465.

[29] C. F. Torres, A. K. Iannillo, A. Gervais, and R. State, "Confuzzius: A data dependency-aware hybrid fuzzer for smart contracts," in *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2021, pp. 103–119.

[30] B. Jiang, Y. Liu, and W. K. Chan, "Contractfuzzer: fuzzing smart contracts for vulnerability detection," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ser. ASE '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 259–269.

[31] C. S. Yashavant, M. Chavda, S. Kumar, A. Karkare, and A. Karmakar, "Scrubd: Smart contracts reentrancy and unhandled exceptions vulnerability dataset," *arXiv preprint arXiv:2412.09935*, 2024.

[32] P. Christen, D. J. Hand, and N. Kirielle, "A review of the f-measure: Its history, properties, criticism, and alternatives," *ACM Comput. Surv.*, vol. 56, no. 3, Oct. 2023.

[33] P. Momeni, Y. Wang, and R. Samavi, "Machine learning model for smart contracts security analysis," in *2019 17th International Conference on Privacy, Security and Trust (PST)*, 2019, pp. 1–6.

[34] 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.

[35] F. Luo, R. Luo, T. Chen, A. Qiao, Z. He, S. Song, Y. Jiang, and S. Li, "Scvhunter: Smart contract vulnerability detection based on heterogeneous graph attention network," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24. New York, NY, USA: Association for Computing Machinery, 2024.