SCStudio: A Secure and Efficient Integrated Development Environment for Smart Contracts

Meng Ren
Tsinghua University
Beijing, China

Fuchen Ma
Tsinghua University
Beijing, China

Zijing Yin
Tsinghua University
Beijing, China

Huizhong Li
WeBank
Shenzhen, China

Ying Fu
Ant Financial
Beijing, China

Ting Chen
University of Electronic Science and Technology of China
Chengdu, China

Yu Jiang*
Tsinghua University
Beijing, China
jiangyu198964@126.com

ABSTRACT

With the increasing popularity of block-chain technologies, more and more engineers use smart contracts for application implementation. Traditional supporting tools can either provide code completions based on static libraries or detect a limited set of vulnerabilities, which results in the manpower waste during coding and miss-detection of bugs. In this work, we propose `SCStudio`, a unified smart contract development platform, which aims to help developers implement more secure smart contracts easily. The core idea is to realize real-time security-reinforced recommendation through pattern-based learning; and to perform security-oriented validation via integrated testing. `SCStudio` was implemented as a plug-in of VS Code. It has been used as the official development tool of WeBank and integrated as the recommended development tool by FISCO-BCOS community. In practice, it outperforms existing contract development environments, such as Remix, improving the average word suggestion accuracy by 30%-60% and helping detect about 25% more vulnerabilities.

The video is presented at https://youtu.be/l6hW3Ds5Tkg.

CCS CONCEPTS

• **Software and its engineering** → **Software testing and debugging**.

KEYWORDS

Smart Contract; Security-Reinforced Code Suggestion; Validation

Yu Jiang* is the corresponding author.

ISSTA '21, July 11–17, 2021, Virtual, Denmark
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8459-9/21/07…$15.00
https://doi.org/10.1145/3460319.3469078

ACM Reference Format:
Meng Ren, Fuchen Ma, Zijing Yin, Huizhong Li, Ying Fu, Ting Chen, and Yu Jiang*. 2021. SCStudio: A Secure and Efficient Integrated Development Environment for Smart Contracts. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '21), July 11–17, 2021, Virtual, Denmark.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3460319.3469078

1 INTRODUCTION

With the growth of application scenarios, smart contracts are attracting more and more users [1, 4, 21]. Different from traditional applications implemented by programming languages such as C and Java, smart contracts written in Solidity often have complex domain-specific business logic and numerous distinctive features, such as the gas system. However, due to the distributed execution environment and the complexity of programming languages, developing secure smart contracts can be challenging even for experienced developers [7]. Existing auxiliary platforms mainly have two pain points. First, most of them only support auto-completion based on fuzzy match, so that they fail to deal with the contextual relevance and lack security considerations of the candidate words. Second, they usually integrated little or no contract validation tools, and need extra configurations of independent analyzers for vulnerability detection, which is extremely unfriendly to users and results in inevitable omission of security vulnerabilities. Therefore, it is necessary to develop a unified platform to help engineers during coding and testing.

In this paper, we propose `SCStudio`[1], an integrated development platform that supports smart contract rapid implementation and comprehensive testing. For security-oriented suggestion, we build a Bidirectional Long and Short Term Memory (BLSTM) network based language model and a context-based word selection algorithm to provide reasonable candidate words, where the contracts crawled down for training are reinforced with domain-specific patch patterns and secure programming standards such as the use of SafeMath libraries. For security-oriented validation, we integrate

[1]https://github.com/FISCO-BCOS/SCStudio

five free and open-source vulnerability detection tools for comprehensive testing. After de-duplicating and merging the test results of each tool, it is able to expose 54 types of vulnerabilities. In practice, we also design a front-end and back-end interaction mode to make the system as lightweight as possible, which frees the engineers from complex configurations.

For evaluation, we collected 47,398 real-world contracts and 181 vulnerable contracts with clear annotations. The results show that SCStudio outperforms existing tools. Compared with state-of-the-art platform (Remix), it improves the average word suggestion accuracy by 30%-60%, and can detect 25% more vulnerabilities. As to actual application in WeBank, a well-known FinTech and blockchain company in China, SCStudio is able to generate and display all candidates within a second, complete security analysis and display the modification suggestions within one minute. Furthermore, it has been integrated as the recommended development toolby FISCO-BCOS community.

2 RELATED WORK

Code Completion. Code completion is one of the most common program automation technologies and is essential to improve programming efficiency. In recent years, many works that explore the application of statistical learning and sequence models focus on the code completion task. For example, the probabilistic modeling of code token sequences proposed by Hindle et al. [10] in 2012. Other works apply probabilistic grammars [2, 5] to the code completion task. Li et al. [13] and Liu et al. [14] use the AST sequence to predict the terminal and non-terminal nodes of the program. In 2016, Raychev et al. [18] used decision tree to directly model the tree structure of the program to make the predictions.

Smart Contract Audits. Smart contracts have been shown to be exposed to severe vulnerabilities [3, 11], and many efforts have been devoted to ensuring the correctness. Luu et al. [15] designed Oyente, which builds the control-flow graph from the bytecode and then performs symbolic execution and checks dangerous patterns. Zeus [12] is another sound analyzer that translates contracts to the LLVM framework and uses XACML to write properties. Others carried out researches from the perspective of execution. Malicious behavior will be detected and blocked from EVM layer through analyzing opcode sequences, like EVM* [16] and Sereum [19]. Different from above works, our focus is to implement a unified development platform to ensure the security of smart contract code throughout the development process such as coding, debugging and testing.

3 SCSTUDIO DESIGN

The overall workflow of SCStudio is shown in Fig. 1, which consists of two major components, security-reinforced code suggestion module and security-oriented code validation module. Each of them takes the contract code in the editing interface as input. The suggestion module will call a pre-trained language model to predict possible next words according to current context and cursor position, and restore special symbols in the prediction result based on the user-defined information in the context. The validation module will execute the built-in detection tools in parallel and synthesize their output information, then generate a detailed bug report.

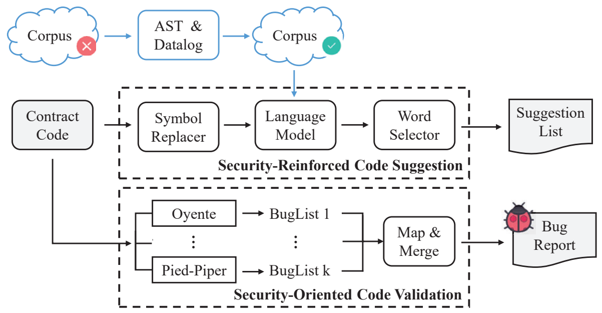


Figure 1: Overall workflow of SCStudio. The blue part is the reinforcement before training, where the red cross subscript represents vulnerable, and the green tick subscript represents secure. The gray shaded part is the input and output.

3.1 Security-Reinforced Code Suggestion

Smart contracts have strict requirements on the logical rigor of the code, reasonable code recommendation with security constraints can effectively prevent embedded vulnerabilities, which lightens the workload of code review and modification. For ensuring the security of the recommended code, the first thing is to construct a dataset that contains reinforced contracts with no security risks, then feed them into the language model.

Domain-specific reinforcement. According to [8, 17], 97% of deployed contracts on Etherscan [9] are vulnerable, which means that, the crawled contracts may violate some programming standards and contain vulnerabilities. Hence, we strengthened the original code based on AST and Datalog respectively. Based on AST, we developed an automatic alteration tool, which traverses AST nodes and replaces the naked binary operations with safe function calls with built-in constraints. Based on Datalog, we defined rules and specific data structures related to different types of vulnerabilities. Then, we performed pattern matching to find vulnerable structures or calling sequences, and accomplished the reinforcement through modifying statements in static single assignment form.

Context-sensitive suggestion. For the language model, some words are of little learning value, such as the name of user-defined variables, functions, parameters and contracts. Therefore, we will traverse the child nodes of each statement, and perform symbol substitution on user-defined information, such as variable names, function names and contract names. Then, we used a tokenizer to realize sub-word division and filter the low-frequency words. Next, we fed these data into a bidirectional LSTM network with attention mechanism. After training, the model can provide a word list which may contain some substituted symbols, then we performed a context-based word selection algorithm to replace these symbols with possible valid words.

3.2 Security-Oriented Code Validation

Integrated security testing can greatly avoid the omission of potential vulnerabilities and reduce unnecessary workload of developers. Combining with static and dynamic analysis, SCStudio can cover the most types of smart contract vulnerabilities. In the process of integration, we first configured the independent environment of each tool and deployed it on the server. Then, we implemented a script to call each tool in parallel through command-line interface. When all tools have finished the execution or reached the preset time, we extracted the vulnerability type, location and other information from the output file to form the final report.

4 USAGE OF SCSTUDIO

4.1 Tool Implementation

We implemented SCStudio as a light-weight plug-in of VS Code. Users can install it directly in the extension store, and then select shortcut instructions starting with "SCStudio" to obtain services. The front-end is implemented in TypeScript, and is mainly responsible for monitoring the interface status, and recording the context and cursor position in current window. In the back-end, SCStudio integrates five free and open-source tools as the sub-detectors, which are Oyente v0.2.7 [15], Mythril-classical v0.22.1 [6], Securify v1.0.0 [25], SmartCheck v2.0.0 [24], and Pied-Piper v1.0.0 [26]. Solc 0.4.24 compiler [23] is used to construct AST files and control-flow graph. All the communication is through HTTP protocol, and the back-end server processes the requests through Flask [20].

4.2 Running Example

As shown in Fig. 2, after starting VS Code, the user can click on the "Extensions" in the left sidebar, type "SCStudio" in the search box and select SCStudio v1.0.0 to install. When the installation is complete, follow the prompts of VS Code to reload the window.

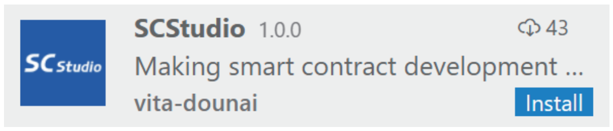
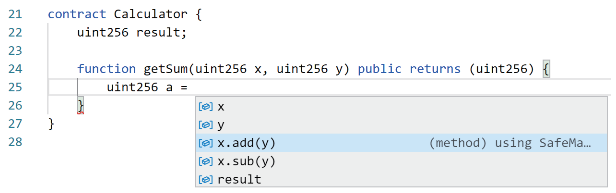


Figure 2: The research result in the extension marketplace.

Then, click File → Preferences → Settings → Extensions → SCStudio in the menu bar to configure. ***Max Waiting Time*** is the maximum timeout for security analysis, which can prevent SCStudio from waiting endlessly in time-consuming operations such as network interaction. The default value is 60. ***Server Address*** is used to specify the local server address of the back-end service, including the IP address and its port, to serve users who have higher requirements for privacy. When this item is empty, the contract code will be submitted to our trial server.

When the user creates or opens a file with the extension ".sol", SCStudio will be automatically loaded and initialized, then starts to monitor the interface status. "Space" is the trigger signal for the recommended service, and a list of possible next-words with security constraints will be displayed on the screen. Considering the situation in Fig. 3, when the developer declares two variables, one of his next possible operations is to get the sum. In the list of recommended tokens, the corresponding one is $x.add(y)$. SCStudio calls the *add* function in the SafeMath library to implement the sum operation, avoiding the integer overflow vulnerability.

The user can call SCStudio to check the security of the contract code in current editing window with command "SCStudio: Analyze Contract" or "Ctrl+F10" shortcut keys. When an issue is detected, SCStudio will give an explicit reminder in the form of a ribbon.

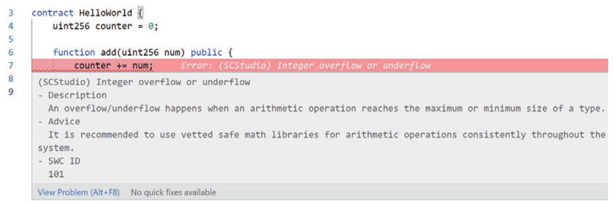


```
contract Calculator {
    uint256 result;

    function getSum(uint256 x, uint256 y) public returns (uint256) {
        uint256 a =
    }
}
```

Figure 3: Example of predictions with security constraints.

Hovering the mouse on the ribbon will display detailed descriptions, repair suggestions and other information, as shown in Fig. 4.

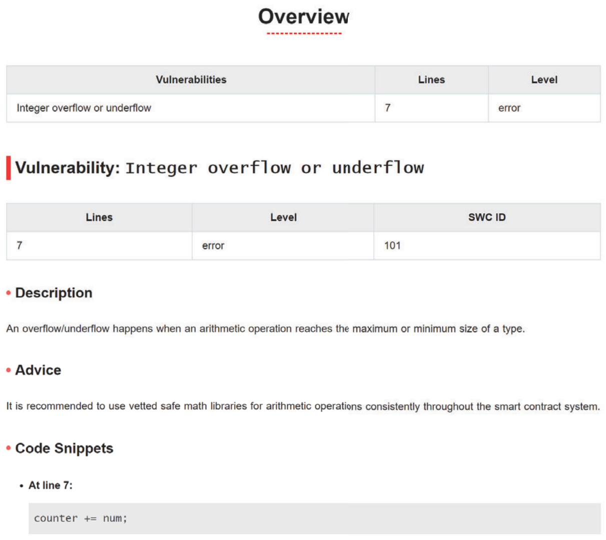


```
3  contract HelloWorld {
4      uint256 counter = 0;
5
6      function add(uint256 num) public {
7          counter += num;        Error: (SCStudio) Integer overflow or underflow
8
9
```

```
(SCStudio) Integer overflow or underflow
- Description
  An overflow/underflow happens when an arithmetic operation reaches the maximum or minimum size of a type.
- Advice
  It is recommended to use vetted safe math libraries for arithmetic operations consistently throughout the
system.
- SWC ID
  101
```

Figure 4: Example of integer overflow detection.

Furthermore, a report in HTML format will be generated and saved to the local directory, which provides the information of all potential vulnerabilities found in the contract, as shown in Fig. 5.



Overview

Vulnerabilities	Lines	Level
Integer overflow or underflow	7	error

Vulnerability: Integer overflow or underflow

Lines	Level	SWC ID
7	error	101

- **Description**

An overflow/underflow happens when an arithmetic operation reaches the maximum or minimum size of a type.

- **Advice**

It is recommended to use vetted safe math libraries for arithmetic operations consistently throughout the smart contract system.

- **Code Snippets**
  - **At line 7:**

```
counter += num;
```

Figure 5: Detailed bug report in HTML format.

5 PRELIMINARY EVALUATION

We evaluated SCStudio with state-of-the-art platform (Remix). This section shows some preliminary evaluation results. We built a corpus by collecting 47,398 unique real-world smart contracts crawled from Ethereum network [9], and a test contract set of size 181 with clear annotations, which consists of 131 vulnerable contracts with 176 tagged vulnerability labels collected by SmartBugs [22] and 50 vulnerable contracts with backdoor labels. All experiments were performed atop a machine with 8 cores (Intel i7-7700HQ @3.6GHz), 24GB of memory, and Ubuntu 16.04.6 as the host operating system.

Accuracy of Code Suggestion. To verify `SCStudio`'s performance on code suggestion, we randomly selected 10 Ethereum projects (108 contracts) from the test corpus and carried out experiments compared with Remix. The results are shown in Table 1, where *Top-K* means the prediction matches the expected answer in *K* candidates. Consider Top-1 accuracy, in about 71.26% cases, users can find their expected next-word on the top of the list, that is 5X higher than Remix's. For Top-5 accuracy, the performance can be improved by 47.12% and achieves an average accuracy of 83.26%. When we expand the number of candidates to 10, the accuracy will rise up to 81.96%-96.62%, which is 41% higher than Remix.

Table 1: Accuracy of security-reinforced code suggestion.

Project	Top-1		Top-5		Top-10	
	Remix	SCStudio	Remix	SCStudio	Remix	SCStudio
Airdrop	9.52%	63.64%	42.86%	77.62%	57.14%	86.52%
ARIYAX	14.29%	77.27%	28.57%	88.64%	38.10%	93.77%
CrystalDeposit	4.76%	70.68%	33.33%	79.50%	33.33%	85.88%
EthVentures4	9.09%	53.99%	22.73%	74.44%	40.91%	81.96%
Eximchain	13.64%	69.57%	45.45%	84.01%	54.55%	91.87%
GasManager	21.74%	73.97%	43.48%	82.19%	60.87%	89.56%
Ipsx	8.70%	78.15%	34.78%	88.74%	52.17%	94.92%
KyberNetwork	19.05%	80.87%	23.81%	91.28%	42.86%	96.62%
OneEight	22.73%	75.98%	36.36%	84.08%	50.00%	93.11%
UpgradeProxy	18.18%	68.46%	50.00%	82.08%	63.64%	88.95%
Average	14.17%	71.26%	36.14%	83.26%	49.36%	90.32%

Effectiveness of Integrated Validation. For code validation, we compared `SCStudio` with other contract auditing tools and Remix, whose core is Mythril. Table 2 summarizes the results. When developers only use one tool, at most 56.6% of hidden vulnerabilities can be detected. With integrated testing, `SCStudio` can cover all categories of vulnerabilities, and successfully exposed 81.9% of hidden security issues. With `SCStudio`, developers do not need to perform unnecessary operations, such as tool switching, environment configuration and manual screening of duplicate alarms.

Table 2: Vulnerabilities detected by Remix and `SCStudio`.

Category	Files	Vulns	Remix	SCStudio
Access Control	17	19	4 (21%)	4 (21%)
Arithmetic	15	24	14 (58.3%)	17 (70.8%)
Backdoor Threats	50	50	0 (0%)	48 (96%)
Front Running	4	7	4 (57.1%)	4 (57.1%)
Locked Ether	2	2	1 (50%)	2 (100%)
Reentrancy	31	32	29 (90.6%)	29 (90.6%)
Timestamp Dependency	5	7	0 (0%)	2 (28.6%)
Unchecked Low Calls	53	78	72 (92.3%)	72 (92.3%)
Unhandled Exception	4	7	4 (57.1%)	7 (100%)
Total	181	226	128 (56.6%)	185 (81.9%)

6 CONCLUSION

In this paper, we proposed `SCStudio`, an integrated smart contract development platform which aims to help developers write smart contracts quickly and securely. First, we conducted a domain-specific reinforcement for original contracts and designed the first context-sensitive code suggestion model for Solidity language. Then, we realized the security-oriented code validation by integrating five advanced tools, so that it can cover 54 types of common issues and provide targeted modification suggestions automatically. Currently, `SCStudio` has been listed as the official smart contract development tool of WeBank. In practice, it outperforms the best development environment in terms of prediction accuracy and vulnerability coverage, and has been integrated as the recommended development tool by FISCO-BCOS community.

ACKNOWLEDGEMENT

This research is sponsored in part by the NSFC Program (No. 62022046, U1911401, 61802223), National Key Research and Development Project (Grant No. 2019YFB1706200), the Webank-Tsinghua Smart Contract Security-Assured Platform Research Project (No. 20202000347).

REFERENCES

[1] Maher Alharby and Aad Van Moorsel. 2017. Blockchain-based smart contracts: A systematic mapping study. *arXiv preprint arXiv:1710.06372* (2017).

[2] Miltiadis Allamanis and Charles A. Sutton. 2014. Mining idioms from source code. *ArXiv* abs/1404.0417 (2014).

[3] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. 2016. A survey of attacks on Ethereum smart contracts. *IACR Cryptology ePrint Archive* 2016 (2016), 1007.

[4] Massimo Bartoletti and Livio Pompianu. 2017. An empirical analysis of smart contracts: platforms, applications, and design patterns. In *International conference on financial cryptography and data security*. Springer, 494–509.

[5] Pavol Bielik, Veselin Raychev, and Martin T. Vechev. 2016. PHOG: Probabilistic Model for Code. In *ICML*.

[6] ConsenSys. 2019. Mythril. https://github.com/ConsenSys/mythril-classic.

[7] Kevin Delmolino, Mitchell Arnett, Ahmed Kosba, Andrew Miller, and Elaine Shi. 2015. Lab: Step by Step towards Programming a Safe Smart Contract. (2015).

[8] Thomas Durieux, João F. Ferreira, Rui Abreu, and Pedro Cruz. 2019. Empirical Review of Automated Analysis Tools on 47,587 Ethereum Smart Contracts. arXiv:1910.10601 [cs.SE]

[9] Etherscan. 2019. Etherscan. https://etherscan.io/.

[10] Abram Hindle, Earl T. Barr, Zhendong Su, Mark Gabel, and Premkumar T. Devanbu. 2012. On the naturalness of software. In *ICSE 2012*.

[11] Yoichi Hirai. 2016. Formal verification of Deed contract in Ethereum name service. *November-2016.[Online]. Available: https://yoichihirai. com/deed. pdf* (2016).

[12] Sukrit Kalra, Seep Goel, Mohan Dhawan, and Subodh Sharma. 2018. ZEUS: Analyzing Safety of Smart Contracts. In *NDSS*.

[13] Jian Li, Yue Wang, Michael R. Lyu, and Irwin King. 2018. Code Completion with Neural Attention and Pointer Networks. *ArXiv* abs/1711.09573 (2018).

[14] Chang Liu, Xin Wang, Richard Shin, Joseph E. Gonzalez, and Dawn Song. 2017. Neural Code Completion.

[15] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. 2016. Making Smart Contracts Smarter. *IACR Cryptology ePrint Archive* 2016 (2016), 633.

[16] F. Ma, Y. Fu, M. Ren, M. Wang, Y. Jiang, K. Zhang, H. Li, and X. Shi. 2019. EVM*: From Offline Detection to Online Reinforcement for Ethereum Virtual Machine. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 554–558.

[17] Ivica Nikolic, Aashish Kolluri, Ilya Sergey, Prateek Saxena, and Aquinas Hobor. 2018. Finding The Greedy, Prodigal, and Suicidal Contracts at Scale. *CoRR* abs/1802.06038 (2018).

[18] Veselin Raychev, Martin T. Vechev, and Eran Yahav. 2014. Code completion with statistical language models. In *PLDI '14*.

[19] Michael Rodler, Wenting Li, Ghassan O Karame, and Lucas Davi. 2018. Sereum: Protecting Existing Smart Contracts Against Re-Entrancy Attacks. (2018).

[20] Armin ronacher. 2010. Flask. https://flask.palletsprojects.com/en/1.1.x/.

[21] Sara Rouhani and Ralph Deters. 2019. Security, performance, and applications of smart contracts: A systematic survey. *IEEE Access* 7 (2019), 50759–50779.

[22] smartbugs. 2020. smart contracts dataset. https://github.com/smartbugs/smartbugs-wild.

[23] Solidity. 2018. Solidity Programming Language. https://git.io/vFA47/.

[24] Sergei Tikhomirov, Ekaterina Voskresenskaya, Ivan Ivanitskiy, Ramil Takhaviev, and Yaroslav Alexandrov. 2018. SmartCheck: static analysis of ethereum smart contracts. In *the 1st International Workshop*.

[25] Petar Tsankov, Andrei Marian Dan, Dana Drachsler-Cohen, Arthur Gervais, Florian Buenzli, and Martin T. Vechev. 2018. Securify: Practical Security Analysis of Smart Contracts. In *ACM Conference on Computer and Communications Security*.

[26] Tsinghua University. 2019. Pied-Piper: Revealing the Backdoor Threats in Smart Contracts. https://github.com/renardbebe/BackdoorDetector.