*Fuzzy Systems and Data Mining IX*
*A.J. Tallón-Ballesteros and R. Beltrán-Barba (Eds.)*

*doi:10.3233/FAIA231090*

# Power of Fuzzing and Machine Learning in Smart Contract Security Validation

Ghazi Mergani Ali[a,1] and Hongsong Chen [a]
[a] *University of Science and Technology Beijing, China*

**Abstract.** Blockchain technology revolutionized digital payments and transactions by introducing disruptive capabilities. It operates through smart contracts, automated software code facilitating transactions without intermediaries. Smart contracts, written in various languages, have gained popularity but are vulnerable to logic flaws and security threats, potentially leading to financial losses and undermining blockchain integrity. Validating their security is essential, although current tools only detect specific attacks, lacking a comprehensive automated solution. This article presents a two-stage process wherein the initial phase involves the extraction of characteristics from smart contracts through the analysis of abstract syntax trees (ASTs) and control flow graphs (CFGs). In order to improve the precision of our results, we can leverage the distinctive capabilities of the fuzzing technique to label the data prior to its integration into the training model. In the subsequent phase, approach is utilized neural decision tree (NDT) typically combines a decision tree structure with neural network components. The mixed-method approach is utilized, leveraging the distinct advantages offered by neural networks and decision trees. The purpose of this collaborative technique is to enhance the probability of identifying vulnerabilities in smart contracts. The experimental assessment yielded favorable outcomes in the detection of smart contract vulnerabilities, namely those pertaining to Reentrancy, integer overflow, and Block Number Dependency.

**Keywords.** Blockchain, smart contract, fuzzing tools, machine learning, neural decision trees

## 1. Introduction

In the age where blockchain technologies are flourishing, smart contracts have become a fundamental element in decentralized applications, bestowing automation, transparency, and security to a range of transactions and engagements. Smart contracts are contracts that execute themselves, with the conditions embedded directly into the code [1]. These contracts serve as the foundational elements of decentralized applications and are crucial for executing intricate business processes on the blockchain. Nevertheless, the attributes that make smart contracts so vital, such as their unchangeable nature and independence, also make them vulnerable to harmful manipulations if they are not meticulously crafted.

[1] Corresponding author: Department of Computer Science and Technology, University of Science and Technology Beijing (USTB), Beijing 100083, China. E-mail: ghazi_en87@yahoo.com.
[2] Second author: a professor in Department of Computer Science, University of Science and Technology Beijing (USTB), Beijing 100083, China. E-mail: chenhs@ustb.edu.cn

The escalating emergence and deployment of smart contracts have concurrently revealed an abundance of security impediments, with vulnerabilities being at the forefront. The sensitivities of these contracts, particularly due to their extensive integration with digital assets, bear the risk of engendering significant financial detriments. These substantial setbacks have spurred significant attention towards the imperative task of safeguarding and ensuring the stability of smart contracts. A scrutiny of existing literature evinces that a multitude of efforts have been initiated to tackle the security conundrums associated with smart contracts [2].

Several studies have delved into the exploration of methods for detecting vulnerabilities in smart contracts. One approach, formal verification, has encountered various challenges. It tends to be susceptible to errors due to its reliance on artificial rules, and it struggles to encompass intricate patterns comprehensively. Additionally, the comparative analysis in this method often necessitates the use of complex symbols and contrast indicators, demanding substantial computational resources and leading to prolonged detection times.

This paper presents a machine learning model for detecting vulnerabilities in smart contracts, utilizing over 1100 verified contracts. The approach involves a two-step process: Preparation and Training. During Preparation, meaningful attributes are extracted from contract code, including static features from abstract syntax trees (ASTs) and control flow graphs (CFGs). Then we used established vulnerability detection tools such as sFuzz [3], CONFUZZIUS [4], and xFuzz [5] to label our training data. In the Training Model step, a mixed-method approach combines decision trees and neural networks, specifically employing the Neural Decision Trees(NDT) model. This model combines decision tree-based partitioning with neural networks to capture complex data relationships, offering both accuracy and interpretability. To summarize, our main contributions are as follows:

1. We propose a model for automated detection of vulnerabilities in Ethereum smart contracts using neural decision trees.
2. It differs from the existing works in that for the first time in our knowledge, labeling data by using the fuzzing tools (xFuzz, sFuzz, CONFUSS) for more accuracy.
3. The efficacy of the model has been demonstrated in terms of its suitability for identifying vulnerabilities in smart contracts. The model was executed on authentic contracts, resulting in a prediction recall and precision of 92%. Furthermore, the model has a detection time of around 5 seconds per contract.

## 2. Background

We briefly review smart contracts vulnerabilities detection in Section 2.1, and machine learning vulnerability detection in smart contracts in Section 2.2, and Neural Decision Trees in Section 2.3

### *2.1. Smart Contracts Vulnerability Detection Methods*

The ascendancy of Blockchain technology and the concomitant prominence of smart contracts have engendered a burgeoning interest in smart contract security. Consequent to the high stakes often implicated by smart contracts, particularly in financial applications, the detection and amelioration of vulnerabilities have become an area of critical import. In this section, we will introduce three kinds of smart contract

vulnerability detection methods, including Static analysis, Dynamic analysis, Fuzzing Method.

**Static analysis** is a technique that can be employed to scrutinize Solidity smart contracts without the need for their execution[6]. This methodology involves examining the Solidity source code of the smart contract to identify potential vulnerabilities and verify adherence to coding standards [7]. Several tools have been developed for static analysis, Slither [8], is a tool that requires access to the source code of the smart contract. It's mainly used to identify vulnerabilities and issues at the code level, but it may not be as effective in identifying runtime vulnerabilities that only manifest when the contract is executed. Slither can generate false positives.

**Dynamic analysis** is a method that examines a program while it is executing or running [9]. It involves analyzing the program's behavior and interactions with its environment, including the inputs and outputs, to identify vulnerabilities and potential issues that may not be apparent from the source code alone. Dynamic analysis can simulate various scenarios, including the injection of malicious inputs, to uncover vulnerabilities that an attacker might exploit.

**Fuzzing Method** Fuzzing is a software testing approach that is automated, wherein potential data is supplied as inputs to a program[10]. The process of fuzzing generates partially valid inputs that are not immediately rejected by the parser. However, these inputs do expose corner cases that were inadequately handled in earlier stages of the program. ContractFuzzer [11] is the first smart contract fuzzing tool that creates fuzzing inputs by leveraging the Application Binary Interface (ABI) specifications of smart contracts. It further employs test oracles to find potential security vulnerabilities and utilizes the Ethereum Virtual Machine (EVM) to capture runtime behaviors of smart contracts.

### *2.2. Machine Learning Vulnerability Detection in Smart Contract*

The public's interest in smart contract security has been considerable, which has prompted noteworthy developments in machine learning-based contract vulnerability detection methods. The two steps of current software vulnerability detection methods, which are based on machine learning and cutting-edge techniques, are 1) training and 2) detection. When the prediction model is fully trained, satisfactory results can be obtained. Currently, there is a need to address the issue of integrating ML technologies with smart contract detection methods. A few papers provide an overview of the literature[12] [13] [14] [15] [16] have been proposed and they have been using machine learning models to detect vulnerabilities in smart contract All these deep learning methods papers are trying to extract features from opcode and label contracts with types of vulnerabilities of smart contract to prepare data and employ machine learning algorithms to detect vulnerabilities in smart contracts.

Other papers [17][18][19] have been conducted to identify smart contract vulnerabilities through the application of deep learning methodologies. However, these studies have primarily focused on extracting features directly from the source code of smart contracts in order to construct a training dataset based on the Abstract Syntax Tree (AST) representation. This approach is advantageous as it allows for the utilization of high-level programming languages and facilitates the processing of the AST. The utilization of static code analyzers for the purpose of extracting features and assigning labels to the smart contract. After the preparation of a training dataset, deep learning models can be trained on it in order to classify new data and identify vulnerabilities. All

existing machine-learning papers are trying to label dataset by using Symbolic Execution or Static analysis methods to detect potential vulnerabilities in the contracts.

### *2.3. Neural Decision Trees (NDT)*

Neural Decision Trees be used for classification tasks. NDT are hybrid machine learning models that combine the strengths of decision trees and neural networks [20]. The primary concept of Neural Decision Trees (NDT) is employing a neural network to reduce interdependencies among input variables initially, followed by feeding the changed input variables into a decision tree learning process for the purpose of categorization. Therefore, the NDT algorithm commences by accepting training data as input for the neural network model. The subsequent output is subsequently forwarded for decision reconstruction, and the resulting rules will be the final output of the NDT model. The NDT model can be applied to predict by combining decision tree-based partitioning with neural networks for capturing complex relationships in the data. It offers the advantage of interpretability while delivering accurate predictions [21].

## 3. Overview Our Method

The proposed model for detecting smart contract vulnerabilities in machine learning leverages the utilization of multiple fuzzing analyzers, thereby optimizing the accuracy. To accomplish this, a collection of machine learning classifiers is trained using established vulnerabilities. The objective is to forecast the presence of similar security vulnerabilities in a novel contract. There are two main steps to creating our machine learning model vulnerability detection in smart contracts the first step is machine learning preparation and the second step is the machine learning training model. The Proposed Architecture in Figure 1. A comprehensive elucidation of each individual phase is included in the subsequent sections.

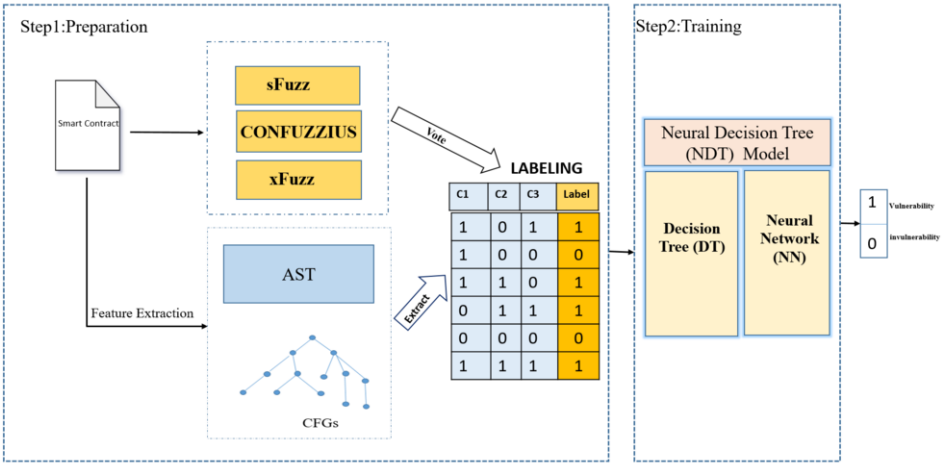


**Figure 1**. Proposed Architecture

### *3.1. Step1: Machine Learning Preparation*

In this section, we provide a detailed explanation of the preparation process employed for training the machine learning model, which is designed to identify vulnerabilities present in smart contracts. We discuss the data collection in Section 3.1.1 and introduce Feature Extraction in Section 3.1.2, Labeling data in Section 3.1.3

#### *3.1.1. Data Collection and Preprocessing*

Supervised learning necessitates a dataset that is adequately large and labeled, while blockchain systems commonly serve as hosts for publicly accessible smart contracts. The decision was made to download and compile contracts via Etherscan, a well-known Ethereum service platform [22]. The data sets at our disposal comprise contracts that exhibit three distinct types of vulnerabilities, namely Reentrancy, Integer Overflow, and Block Number Dependency. In order to ensure representativeness, a comprehensive dataset comprising a total of 1100 contracts was gathered for subsequent analysis. The dataset was partitioned into an 80% training set and a 20% testing set.

#### *3.1.2. Building AST and Feature Extraction*

In preparing the training dataset, features are directly extracted from the smart contract's source code by utilizing a Solidity parser [23] to generate an abstract syntax tree (AST). The data structure of AST is chosen because it contains the abstract syntactic structure and content-related details. Solidity parser is an open-source tool that is relatively simple to install and operate. It examines the syntax of Solidity code and constructs an AST, which can be navigated using the parser's built-in functions and dictionary methods. Moreover, The CFG is employed to extract an additional set of features a graph representation of the different paths a program can take during execution [24]. CFGs provide a visual representation of the program's control flow and are instrumental in extracting information regarding the different execution paths within a program. we use a tool specifically designed for analyzing smart contract Slither (8). This tool provides features extracted from CFGs and can help to identify potential issues and opportunities for improvement in smart contract code. features extracted are 20, of which 13 are extracted from ASTs and the other 7 Features from CFGs

#### *3.1.3. Labeling*

The benefit of using fuzzing tools instead of a particular static tool is that fuzzing tools can reduce false negatives Thus, it increases the accuracy of vulnerability detection We chose the fuzzing tools sFuzz (3), CONFUZZIUS (4), and xFuzz (5). because they detect smart contract vulnerabilities with greater accuracy than other fuzzing tools. In this process, we chose these tools that could detect a variety of common smart contract vulnerabilities. In the labeling process, vulnerability reports from various detection tools are collected for each contract by taking the common vulnerabilities (Reentrancy, Integer Overflow, Block Number Dependency), and the value of each label is either 0 or 1. When the value is 1, it indicates that the contract does have a vulnerability of that sort. and when the value is 0, it indicates that the contract does not have a vulnerability, a contract is labeled as vulnerable (1) if it secures at least two votes in the analysis of fuzzing tools indicating a vulnerability. otherwise, the labels as non-vulnerable (0). the quality of the

labels and features will directly impact the performance of the machine learning models, so it's important to ensure accuracy in this labeling process.

### *3.2. Step2: Machine Learning Training Model*

We take the features from smart contracts and use a neural decision tree (NDT) model to train three types of vulnerability. a neural diction tree network was trained with the following parameters:

1) Number of Trees: 10
2) Depth: 10
3) Feature Rate: 1.0
4) Number of Classes:3
5) Number of Features: 11

The core of the algorithm involves building and training a Neural Decision Tree (NDT) model, which is a hybrid of neural networks and decision trees. Once the model is trained, it can predict class labels (0 or 1) for each data point in the testing set. The algorithm also evaluates the model's performance using metrics such as accuracy, precision, recall, and F1-score, and it generates a confusion matrix to assess how well the model identifies normal and vulnerable contracts.

## 4. Experimental Evaluation

In this section, we first present the metrics used for the model evaluation. Then we introduce the evaluation results of the implications of using our (NDT) model compared with the decision tree (DT) on security vulnerability analysis on smart contracts.

### *4.1. Evaluation Metrics*

The proposed work employs a set of metrics including accuracy, precision, recall, F1 score, and the confusion matrix to assess the performance of learning algorithms for classification (NDT), with the choice of metrics playing a pivotal role in model selection. Specifically, the study primarily focuses on accuracy as the predominant metric for evaluating classification models, defined as the ratio of correctly predicted instances to all predictions and calculated through the confusion matrix components (False Negatives, True Negatives, True Positives, and False Positives). In the context of vulnerability detection, the work emphasizes the significance of recall, precision, and F1-score as critical indicators. Additionally, the evaluation includes accuracy and loss values during the training process, with a preference for minimizing false negatives to detect all potential vulnerabilities and minimizing false positives to enhance the effectiveness of the analysis. As a result, the F1-score emerges as a reliable measure, accommodating imbalanced data, while performance evaluation relies on precision, recall, and the F1 measure computed from the confusion matrix metrics.

### *4.2. Evaluation Results*

Obviously, the performance of (NDT) is the best compared with the decision tree (DT) model (see Table 1) in the sense that all of its indicators are the highest. When comparing

the results of the vulnerability detection models, we observe variations in accuracy, recall, F1-score, and precision across different types of vulnerabilities. We report the experimental results in Table 1.

The findings of our study indicate that the traits we retrieved have the capability to identify vulnerabilities with a satisfactory level of accuracy. Hence, it is postulated that both the structural patterns of code and its complexity have an influence on the presence of vulnerabilities. It has been observed that the detection accuracies of the three vulnerabilities exceed 85%. with other evaluation metrics consistently surpassing the 90% threshold. In the context of the Block Number Dependency vulnerability, it is seen that the NDT model exhibits an accuracy rate of 92.73%, whilst the DT model demonstrates an accuracy rate of 85.45%. The NDT model exhibits superior performance in terms of recall (94%), F1-score (95%), and precision (96%) compared to the DT model in relation to this particular vulnerability. It is worth noting that the NDT model exhibits a slightly lower accuracy index in the case of Integer Overflow, potentially stemming from the multifaceted nature of this vulnerability, where the extracted features may not capture all relevant aspects.

**Table 1** experimental results of three vulnerabilities in terms of accuracy, recall, precision, F1-score and precision

| Vulnerability | Model | Accuracy | Recall | F1-score | Precision |
|---|---|---|---|---|---|
| Reentrancy | NDT | 86.82% | 93% | 91% | 89% |
| | DT | 84.55% | 91% | 90 % | 88% |
| Integer Overflow | NDT | 84.55% | 94% | 90% | 86% |
| | DT | 80.91% | 98% | 88% | 81% |
| Block Number Dependency | NDT | 92.73% | 94% | 95% | 96% |
| | DT | 85.45% | 86% | 90% | 95% |

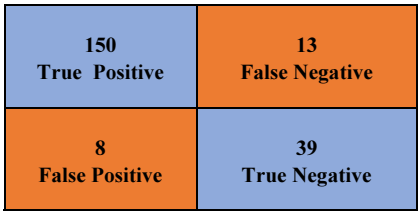


**Figure 2:** NDT Confusion Matrix

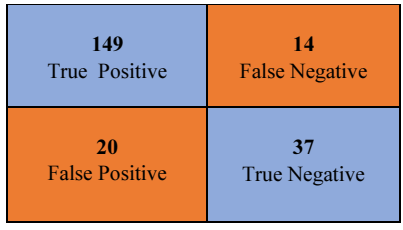


**Figure 3:** DT Confusion Matrix

The confusion matrix, as depicted in Figure 2 and Figure 3, reveals that the NDT algorithm exhibits a lower number of False Positive instances compared to the NT algorithm. This observation suggests that the prediction methodology employed by NDT is effective. The lower value seen in the Precision index of the two models may be attributed to the higher likelihood of false positives associated with Integer Overflow vulnerabilities. The presence of numerous valid operations can lead to integer overflows.

## 5. Conclusion & future work

In this article, we introduce an innovative system for automating the detection of vulnerabilities in Ethereum smart contracts, leveraging machine learning algorithms. Our approach stands out as it pioneers the use of the fuzzing method for data labeling, aiming to enhance accuracy. The model's effectiveness is substantiated by its time-saving capabilities and its proficiency in identifying vulnerabilities within smart contracts. We conducted experiments on real contracts, resulting in an impressive prediction recall and precision rate of 92%. Additionally, the model demonstrates a remarkable detection time of approximately 5 seconds per contract. This article also presents a two-step method strategy, with the first step involving feature extraction from smart contracts using abstract syntax trees (AST) and control flow graphs (CFGs), in the second step by a mixed-method approach combining neural networks and decision trees to data set training. The model we developed exhibited a notable enhancement in both accuracy and efficiency when compared to the direct utilization of fuzzing techniques.

## References


[1] Kerikmäe T, Rull A. Smart Contracts. Futur Law ETechnologies 2016:1–233. https://doi.org/10.1007/978-3-319-26896-5.

[2] Wohrer M, Zdun U. Smart contracts: Security patterns in the ethereum ecosystem and solidity. 2018 IEEE 1st Int Work Blockchain Oriented Softw Eng IWBOSE 2018 - Proc 2018;2018-Janua:2–8. https://doi.org/10.1109/IWBOSE.2018.8327565.

[3] Nguyen TD, Pham LH, Sun J, Lin Y, Minh QT. Sfuzz: An efficient adaptive fuzzer for solidity smart contracts. Proc - Int Conf Softw Eng 2020:778–88. https://doi.org/10.1145/3377811.3380334.

[4] Torres CF, Iannillo AK, Gervais A, State R. Towards Smart Hybrid Fuzzing for Smart Contracts. Proc - 2021 IEEE Eur Symp Secur Privacy, Euro S P 2021 2021:103–19. https://doi.org/10.1109/EuroSP51992.2021.00018.

[5] Xue Y, Ye J, Zhang W, Sun J, Ma L, Wang H, et al. xFuzz: Machine Learning Guided Cross-Contract Fuzzing. IEEE Trans Dependable Secur Comput 2022:1–14. https://doi.org/10.1109/TDSC.2022.3182373.

[6] Cousot P, Cousot R. Abstract interpretation: "A" unified lattice model for static analysis of programs by construction or approximation of fixpoints. Conf Rec Annu ACM Symp Princ Program Lang 1977;Part F1307:238–52. https://doi.org/10.1145/512950.512973.

[7] Harer JA, Kim LY, Russell RL, Ozdemir O, Kosta LR, Rangamani A, et al. Automated software vulnerability detection with machine learning 2018.

[8] Feist J, Grieco G, Groce A. Slither: A static analysis framework for smart contracts. Proc - 2019 IEEE/ACM 2nd Int Work Emerg Trends Softw Eng Blockchain, WETSEB 2019 2019:8–15. https://doi.org/10.1109/WETSEB.2019.00008.

[9] Praitheeshan P, Pan L, Yu J, Liu J, Doss R. Security Analysis Methods on Ethereum Smart Contract Vulnerabilities: A Survey 2019.

[10] Liu C, Liu H, Cao Z, Chen Z, Chen B, Roscoe B. ReGuard: Finding reentrancy bugs in smart contracts. Proc - Int Conf Softw Eng 2018:65–8. https://doi.org/10.1145/3183440.3183495.

[11] Jiang B, Liu Y, Chan WK. ContractFuzzer: Fuzzing smart contracts for vulnerability detection. ASE 2018 - Proc 33rd ACM/IEEE Int Conf Autom Softw Eng 2018:259–69. https://doi.org/10.1145/3238147.3238177.

[12] Wang W, Song J, Xu G, Li Y, Wang H, Su C. ContractWard: Automated Vulnerability Detection Models for Ethereum Smart Contracts. IEEE Trans Netw Sci Eng 2021;8:1133–44. https://doi.org/10.1109/TNSE.2020.2968505.

[13] Xing C, Chen Z, Chen L, Guo X, Zheng Z, Li J. A new scheme of vulnerability analysis in smart contract with machine learning. Wirel Networks 2020. https://doi.org/10.1007/s11276-020-02379-z.

[14] Gopali S, Khan ZA, Chhetri B, Karki B, Namin AS. Vulnerability Detection in Smart Contracts Using Deep Learning. Proc. - 2022 IEEE 46th Annu. Comput. Software, Appl. Conf. COMPSAC 2022, 2022, p. 1249–55. https://doi.org/10.1109/COMPSAC54236.2022.00197.

[15] Solomon GJ, Zhang P, Liu Y, Brooks R. An Efficient Vulnerability Detection Model for Ethereum

Smart Contracts. Lect Notes Comput Sci (Including Subser Lect Notes Artif Intell Lect Notes Bioinformatics) 2019;11928 LNCS:371–86. https://doi.org/10.1007/978-3-030-36938-5_22.

[16] Wang Z, Wu W, Zeng C, Yao J, Yang Y, Xu H. Graph Neural Networks Enhanced Smart Contract Vulnerability Detection of Educational Blockchain 2023.

[17] Momeni P, Wang Y, Samavi R. Machine Learning Model for Smart Contracts Security Analysis. 2019 17th Int Conf Privacy, Secur Trust PST 2019 - Proc 2019. https://doi.org/10.1109/PST47121.2019.8949045.

[18] Narayana KL, Sathiyamurthy K. Automation and smart materials in detecting smart contracts vulnerabilities in Blockchain using deep learning. Mater Today Proc 2022. https://doi.org/10.1016/j.matpr.2021.04.125.

[19] Zhou Q, Zheng K, Zhang K, Hou L, Wang X. Vulnerability Analysis of Smart Contract for Blockchain-Based IoT Applications: A Machine Learning Approach. IEEE Internet Things J 2022;9:24695–707. https://doi.org/10.1109/JIOT.2022.3196269.

[20] Lee YS, Yen SJ. Neural-based approaches for improving the accuracy of decision trees. Lect Notes Comput Sci (Including Subser Lect Notes Artif Intell Lect Notes Bioinformatics) 2002;2454 LNCS:114–23. https://doi.org/10.1007/3-540-46145-0_12.

[21] Li X, Chan CW, Nguyen HH. Application of the Neural Decision Tree approach for prediction of petroleum production. J Pet Sci Eng 2013;104:11–6. https://doi.org/10.1016/j.petrol.2013.03.018.

[22] Etherscan n.d. https://etherscan.io/ (accessed September 27, 2023).

[23] solidity-parser-antlr n.d. https://github.com/federicobond/solidity-parser-antlr (accessed September 15, 2023).

[24] Huang TH-D. Hunting the Ethereum Smart Contract: Color-inspired Inspection of Potential Attacks 2018.