Hunting the Ethereum Smart Contract: Color-inspired Inspection of Potential Attacks

TonTon Hsien-De Huang
Leopard Mobile Inc. (Cheetah Mobile Taiwan Agency)
Department of Computer Science and Information Engineering, National Cheng Kung University
TonTon@TWMAN.ORG

ABSTRACT
Blockchain and Cryptocurrencies are gaining unprecedented popularity and understanding. Meanwhile, Ethereum is gaining a significant popularity in the blockchain community, mainly due to the fact that it is designed in a way that enables developers to write smart contract and decentralized applications (Dapps). This new paradigm opens the door to many possibilities and opportunities. However, the security of Ethereum smart contracts has not received much attention; several Ethereum smart contracts malfunctioning have recently been reported. Unlike many previous works that have applied static and dynamic analyses to find bugs in smart contracts, we do not attempt to define and extract any features; instead we focus on reducing the expert’s labor costs. We first present a new in-depth analysis of potential attacks methodology and then translate the bytecode of solidity into RGB color code. After that, we transform them to a fixed-sized encoded image. Finally, the encoded image is fed to convolutional neural network (CNN) for automatic feature extraction and learning, detecting compiler bugs of Ethereum smart contract.

CCS CONCEPTS
• Computing methodologies → Transfer learning; Supervised learning by classification; • Applied computing → Digital cash;

KEYWORDS
blockchain; ethereum; smart contract; convolutional neural network

1 INTRODUCTION
Since Satoshi Nakamoto published the article "Bitcoin: A Peer-to-Peer Electronic Cash System" in 2008 [1], and after the official launch of Bitcoin in 2009, technologies such as blockchain and cryptocurrency have attracted attention from academia and industry. At present, the technologies have been applied to many fields such as medical science, economics, Internet of Things [2]. Since the launch of Ethereum (Next Generation Encryption Platform) [3] with Vital contract function proposed by Vitalik Buterin in 2015, lots of attention has been obtained on its dedicated cryptocurrency Ether, smart contract, blockchain and its decentralized Ethereum Virtual Machine (EVM). The main reason is that its design method provides developers with the ability to develop Decentralized apps (Dapps), and thus obtain wider applications. A new application paradigm opens the door to many possibilities and opportunities.

Smart contracts are stored in blocks in order to assist in the negotiation and implementation of the verification contract. The
developed, we can provide developers with early detection of relevant vulnerabilities to repair and reduce smart contracts without deploying source code before deployment to reduce the security issues.

2 RELATED WORK
A very recent study conducted analysis on nearly one million contracts; among them, 34,200 contracts are vulnerable and nearly 4,000 contracts are practically exploitable. Smart contracts can hardly be adjusted once deployed. To achieve secure contracts, the key step is to have a thorough security examination before deployment [12]. Loi Luu et al, considered that it is more accurate than the dynamic analysis if the results of the path-by-path method are inferred by static analysis, especially for Ethereum. The uncertainty and complexity of the blockchain behavior makes it difficult to simulate the execution environment. It is the biggest issue for dynamic analysis. Therefore, they developed an execution tool for symbols, called Oyente, to discover potential vulnerability in Ethereum’s smart contracts[14].

Each symbolic path has a path condition which is a formula over the symbolic inputs built by accumulating constraints which those inputs must satisfy in order for execution to follow that path. A path is infeasible if its path condition is unsatisfiable. Otherwise, the path is feasible. Among 19366 existing Ethereum smart contracts, Oyente flags 8833 of them as vulnerable, including the The DAO which led to a $50 million US dollar loss in June 2016 [13]. IBM Research presents a framework (ZEUS) to verify the correctness and validate the fairness of smart contracts. ZEUS abstract interpretation and symbolic model checking, along with the power of constrained horn clauses to quickly verify contracts for safety [15]. At the same time, they have analyzed almost 22.4K smart contracts and discover 94.6% of contracts (containing cryptocurrency worth more than $0.5 billion) are vulnerable. Microsoft Research, Inria, and Harvard University also have developed a dependent types and monadic effects framework F* (a functional programming language aimed at program verification), and automated queries to statically verify properties on EVM bytecode and Solidity sources [16].

In summary, currently, most of the Ethereum security research still rely on labor-intensive examination and focus only on analyzing the control flow graph (CFG) or symbolic execution of the smart contract, so as to determine whether the program under test is causing malicious transaction behavior. This is insufficient in identifying the security flaws in codes. However, Machine Learning/Deep Learning (ML/DL) already has a wide range of applications, especially in security problems, such as spam filtering, botnet detection, and malware classification. Meanwhile our proposed system is designed particularly for the security examination of contracts with the minimum labor cost.

3 OUR PROPOSED METHODOLOGY
We have developed a color representation for translating the bytecode of solidity into RGB color code and transform them to a fixed-sized encoded image. After that, the encoded image is fed to convolutional neural network for automatic feature extraction and learning (without extracting features from the solidity source code manually in advance.), reducing the expert’s labor costs. Such translation is also featured by the fact that more complex information in the solidity source code can be preserved in the color image with 16777216 colors (each sampling with 24 bit pixels) compared to the gray scale image with only 256 colors (each sampling with 8 bit pixels). With the fully connected network infrastructure of DNN, though it can deal with its large amount of parameters, however, the local receptive fields and shared weights of CNN make it more suited for more complex structure. It not only decreases the amount of parameters, but also reflects the complexity of smart contract, saving the time for huge computation with current method.

3.1 The Core Technology of Our Methodology

![Figure 1: The bytecode of the smart contract example.](image1)

![Figure 2: The solidity2jpg of the smart contract example.](image2)
We now explain further in details. Firstly, taking the smart contract of The DAO (0xBB9bc244D798123fDe783FC1C72d33b8C189413) as an example, its bytecode can be obtained in "https://etherscan.io" (as shown in Fig. 1). After that, we perform translating the bytecode of solidity into RGB color code (eg 606060 = (R:96, G:96, B:96), 405260 = (R:64, G:82, B:96), 00357c = (R: 64, G: 81, B: 96)) and then convert bytecode to rgb color code. After this, we can get color images. Then we input these images to CNN for training the compiler bugs detection model for smart contracts. Fig. 2 is an example of converting the bytecode of solidity to jpg.

According to our experience, we also found that two approaches might be used to escape our smart contract detection:

- Since the traditional filter size of CNN is 3*3 or 5*5, the uncorrelated bytecode might become correlated when we transform smart contract into images. The compiler bugs of smart contract may evade the detection by taking advantage of such a mismatch.
- Meanwhile, smart contract color images are not natural images; instead, they are formed from solidity source code. Thus, the pooling inevitably destroys the contexts and semantics of the malware code, causing the detection inaccuracy.

To address the above two issues, we did many experiments with CNN models (includes AlexNet, GoogleNet, and Inception-v3). We found the characteristics of 1x1 convolution in “Network in Network” [17]. 1x1 convolution is equivalent to cross-channel parametric pooling layer, and this cascaded cross channel parametric pooling structure allows complex and learnable interactions of cross channel information.

3.2 The Architecture of Our Methodology

Fig. 3 is a screenshot of our internal proof-of-concept UI, and Fig. 4 is the result of the passback of our RESTful API (multi-label). A brief description of our system flow chart is shown as follows (steps 1-4 are off-line phases for our internal development and steps 5-6 are online phases, where the user can interact with the system. More specifically, after uploading the bytecode and invokes our RESTful API, the user can obtain the analysis result.):

- Step 1. Crawling the bytecode of smart contract from etherscan by pyspider (including benign and malicious ones) as sample;
- Step 2. Transforming the bytecode of smart contract into RGB color code and transforming them to a fixed-sized encoded image;
- Step 3. After that, the encoded image is fed to convolutional neural network for automatic feature extraction and learning (without extracting features from the solidity source code manually in advance.)
- Step 4. Finally, once the model has been trained and validated, we deploy it on the backend server.
- Step 5. Only provide the bytecode of smart contract, we will transform it into smart contract color image.
- Step 6. After the bytecode of smart contract are all identified, the scanned results will be provided to the users through our visualization tool and public RESTful API.

However, we also found more key problems. Because the gap between the number of normal contract and the number of vulnerable contract samples is very large, and currently there is only 17 Solidity known bugs such as "optimizerStateKnowledgeNotResetForJumpdest", "ArrayAccessCleanHigherOrderBits", and "AncientCompiler" etc., and their gap is also very large. We need to collect those imbalanced data and we need sufficient training data in this domain. It means that a picture will be defined in multiple categories. Therefore, based on our previous research [18], we re-implement Inception-v3 to multi-label classification through Transfer Learning. For example, 0xBcF10Dbc180172d8352BE5b8C814E8f3474 is at the same time with Solidity bugs including SolidFunSelectSelector, DelegateCallReturnValue, ECRecoverMalformedInput and SkipEmptyStringValue.

3.2 The Architecture of Our Methodology

Fig. 3 is a screenshot of our internal proof-of-concept UI, and Fig. 4 is the result of the passback of our RESTful API (multi-label). A brief description of our system flow chart is shown as follows (steps 1-4 are off-line phases for our internal development and steps 5-6 are online phases, where the user can interact with the system. More specifically, after uploading the bytecode and invokes our RESTful API, the user can obtain the analysis result.):

- Step 1. Crawling the bytecode of smart contract from etherscan by pyspider (including benign and malicious ones) as sample;
- Step 2. Transforming the bytecode of smart contract into RGB color code and transforming them to a fixed-sized encoded image;
- Step 3. After that, the encoded image is fed to convolutional neural network for automatic feature extraction and learning (without extracting features from the solidity source code manually in advance.)
- Step 4. Finally, once the model has been trained and validated, we deploy it on the backend server.
- Step 5. Only provide the bytecode of smart contract, we will transform it into smart contract color image.
- Step 6. After the bytecode of smart contract are all identified, the scanned results will be provided to the users through our visualization tool and public RESTful API.
We run it on a 64-bit Ubuntu 14.04, and hardware setting are 128 GB DDR4 2400 RAM and Intel(R) Xeon(R) E5-2620 v4 CPU, NVIDIA TITAN V, TITAN XP and GTX 1080 GPUs; more specifically, the software setting is the nvidia-docker tensorflow:18.04-py2 on NVIDIA cloud. The research results and the data will be found on our website http://R2D2.TWMAN.ORG.

We first crawled from the etherscan.io for the verified contract information from Jan. 2018 to Apr. 2018. The smart contract message http://R2D2.TWMAN.ORG. cloud. The research results and the data will be found on our website http://R2D2.TWMAN.ORG.

After that, we collected from May. 2018 to Jun 2018 from etherscan.io for the verified contract information from Jan. 2018 to Apr. 2018. The smart contract message https://github.com/melonproject/oyente.

According to our data, there are 1800 new smart contract produced on ethereum main net per day. Amongst less than 30% is verified by etherscan. Meanwhile according to the third party blockchain evaluation system, Rating Token (https://ratingtoken.io) and coin-schedule, in 2018, there has been 11.75 billion US dollars financed by ICO projects. One of the key factors for the success of a smart contract for each project is the existence of a loophole; Our goal is to optimize the amount of parameters, network structure, and release automated verification tools and public RESTful API. The above experiment results demonstrate that our proposed system can have accurate security analysis on contracts with very limited labor cost.

On the other hand, since the solidity compile bugs of smart contracts have the characteristics of multi-label, we initially identify if there are compile bugs. If the analysis results are malicious, we use transfer learning for multi-label. After training and inference, the results obtained are shown in Figure 4. Fig. 5 is a screenshot of our public proof-of-concept scan result.

5 CONCLUSION
According to our data, there are 1800 new smart contract produced on ethereum main net per day. Amongst less than 30% is verified by etherscan. Meanwhile according to the third party blockchain evaluation system, Rating Token (https://ratingtoken.io) and coin-schedule, in 2018, there has been 11.75 billion US dollars financed by ICO projects. One of the key factors for the success of a smart contract for each project is the existence of a loophole; Our goal is to optimize the amount of parameters, network structure, and release automated verification tools and public RESTful API. The above experiment results demonstrate that our proposed system can have accurate security analysis on contracts with very limited labor cost.

REFERENCES
[1] S Nakamoto. Bitcoin: A peer-to-peer electronic cash system. http://bitcoin.org/bitcoin.pdf.
[2] X Li, P. Jiang, T. Chen, X. Luo, and Q. Wen, “A survey on the security of blockchain systems,” Future Generation Computer Systems, arXiv: 1802.06993.
[3] Vitalik Buterin, aâ€‘â€‘A Next-Generation Cryptocurrency and Decentralized Application Platform.aâ€‘â€‘A Bitcoin Magazine, [Online]. Available: https://bitcoinmagazine.com/articles/ethereum-next-generation-cryptocurrency-decentralized-application-platform-1390528212/
[4] aâ€‘â€‘A Ethereum, A Virtual Currency, Enables Transactions That Rival Bitcoin,aâ€‘â€‘A Nathaniel Popper for the New York Times, [Online]. Available: https://www.nytimes.com/2016/03/28/business/dealbook/ethereum-a-virtual-currency-enables-transactions-that-rival-bitcoins.html.
[5] aâ€‘â€‘A This Is Your Company on Blockchain,aâ€‘â€‘A Bloomberg Businessweek, [Online]. Available: https://www.bloomberg.com/news/articles/2016-08-25/this-is-your-company-on-blockchain.
[6] aâ€‘â€‘A The great chain of being sure about things,aâ€‘â€‘A The Economist, [Online]. Available: https://www.economist.com/briefing/2015/10/31/the-great-chain-of-being-sure-about-things.
[7] P. J. Pasecki, ‘Gaming Self-Contained Provably Fair Smart Contract Casino,’ 2016, vol. 1, p. 12, 2016.
[8] V. Buterin. Critical update re: Dao vulnerability. https://blog.ethereum.org/2016/06/17/critical-update-re-dao-vulnerability, 2016.
[9] "The DAO smart contract," [Online]. Available: http://etherscan.io/address/0xb91bc244d798123fde783fcc1c72d3bb8c189413#code.
[10] "Gas Economics: Call Stack Depth Limit Errors," [Online]. Available: https://github.com/LeastAuthority/ethereumanalyses/blob/master/GasEcon.md#callstack-depth-limit-errors.
[11] "Breaking the House," [Online]. Available: http://martin.swende.se/blog/Breaking the house.html.
[12] I. Nikolic, A. Kolluri, J. Sergey, P. Saxena, and A. Hobor, “Finding The Greedy, Prodigal, and Suicidal Contracts at Scale,” arXiv:1802.06038, 2018.
[13] L. Lui, D. Chu, H. Olickel, P. Saxena, and A. Hobor, “Making smart contracts smarter,” in CCS. ACM, 2016, pp. 254ÂąV269.
[14] "Oyente: An Analysis Tool for Smart Contracts," [Online]. Available: https://github.com/melonproject/oyente.
[15] S. Kalra, S. Goel, M. Dhawan, and S. Sharma, “Zeus: Analyzing safety of smart contracts,” in 25th ISOC Symposium on Network and Distributed System Security (NDSS’18), 2018.
[16] K. Bhargavan, A. Delignat-Lavaud, C. Fournet, A. Gollamudi, G. Gonthier, N. Koblissi, et al., “Formal Verification of Smart Contracts: Short Paper,” presented at the Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security, Vienna, Austria, 2016.
[17] M. Lin, C. Qiang, and Y. Shuicheng, ‘Network In Network,’ 2nd International Conference on Learning Representations (ICLR), Banff, Canada, 2014.
[18] TonTon H.-D. Huang, Chia-Mu Yu, and Hung-Yu Kao, “R2-D2: Color-Inspired Convolutional Neural Network (CNN)-based Android Malware Detection,” arXiv:1705.04448.