

# EASYFLOW : Keep Ethereum Away From Overflow

Jianbo Gao<sup>1</sup>, Han Liu<sup>2</sup>, Chao Liu<sup>1</sup>, Qingshan Li<sup>1</sup>, Zhi Guan<sup>1</sup>, and Zhong Chen<sup>1</sup>

<sup>1</sup>*School of Electronics Engineering and Computer Science, Peking University, Beijing, China*

<sup>2</sup>*School of Software, Tsinghua University, Beijing, China*

**Abstract**—While Ethereum smart contracts enabled a wide range of blockchain applications, they are extremely vulnerable to different forms of security attacks. Due to the fact that transactions to smart contracts commonly involve cryptocurrency transfer, any successful attacks can lead to money loss or even financial disorder. In this paper, we focus on the overflow attacks in Ethereum, mainly because they are widely rooted in many smart contracts and comparatively easy to exploit. We have developed EASYFLOW, an overflow detector at Ethereum Virtual Machine level. The key insight behind EASYFLOW is a taint analysis based tracking technique to analyze the propagation of involved taints. Specifically, EASYFLOW can not only divide smart contracts into safe contracts, manifested overflows, well-protected overflows and potential overflows, but also automatically generate transactions to trigger potential overflows. In our preliminary evaluation, EASYFLOW managed to find potentially vulnerable Ethereum contracts with little runtime overhead. A demo video of EASYFLOW is at <https://youtu.be/QbUJkQI0L6o>.

**Index Terms**—Ethereum, Overflow Vulnerability, Taint Analysis, Smart Contract

## I. INTRODUCTION

Blockchain and smart contracts have been widely applied and adopted since the decentralized cryptocurrency Bitcoin was first introduced by Nakamoto [1]. Ethereum provides a quasi-Turing-complete virtual machine [2] and allows developers to program smart contracts to implement complex functions. Developers can even issue a new cryptocurrency based on Ethereum with only several hundred lines of source code under an ERC Token Standard. However, smart contracts are vulnerable and hackers are intensely attracted as they are closely related to money. Overflow vulnerabilities of smart contracts are especially easy to be exploited compared to different forms of security attacks.

In April 2018, BecToken was attacked by integer overflow on multiplication, causing an extremely large amount of tokens transferred to malicious accounts and the price of BEC cleared. The vulnerable function is located in `batchTransfer` and the code is shown in Fig. 1. As indicated in line 4, the second parameter `_value` can be any 256-bit integer and the result of multiplying `_value` by `cnt` may overflow. In this particular transaction, `_value` was set to `0x8000...000` (63 0s) and `_receivers` were two different addresses, thus the result overflowed and amount was calculated to be 0. Each of these two receivers received about 5.79E58 BECs but 0 BEC was deducted from the sender's account.

Although overflow has become one of the most devastating vulnerabilities, there are few effective detection schemes and

```

1 function batchTransfer(address[] _receivers, uint256
2   _value) public whenNotPaused returns (bool) {
3   uint cnt = _receivers.length;
4   uint256 amount = uint256(cnt) * _value;
5   require(cnt > 0 && cnt <= 20);
6   require(_value>0 && balances[msg.sender]>=amount);
7   ...
8   return true;
9 }
```

Fig. 1: A vulnerable function of BEC

tools. From the practical perspective, the main challenges of detecting overflow vulnerabilities are as follows.

### Challenge 1: Infer and Trigger Potential Overflows.

Overflow only occurs in certain transactions with special input data and message value. Traditional testing techniques are insufficient to infer and trigger potential overflow vulnerabilities in attack-free transactions and have difficulty generating test cases because StateDB must be considered in the Ethereum setting, which consists of account addresses, balances, global variable values.

### Challenge 2: Identify Protection Patterns.

Experienced developers may implement effective protection schemes such as SafeMath library and assertions to protect their contracts from overflow, which may lead to a significant increase in False Positive in the detection tools.

### Our Insight.

To address the challenges, we developed EASYFLOW for detecting overflow vulnerabilities in smart contracts. The key insight behind EASYFLOW is a taint analysis based tracking technique to analyze the propagation of involved taints. In the detection, EASYFLOW monitors the transaction process, captures manifested overflows, identifies well-protected overflows, and automatically generates transactions to trigger potential overflows. We have applied EASYFLOW to detect real transactions on Ethereum Mainnet, and found it efficient to discover vulnerable contracts even from attack-free transactions.

## II. OVERVIEW

The general work flow of EASYFLOW is shown in Fig. 2. Specifically, EASYFLOW consists of four components in high-level design. It takes transactions as input, and StateDB is accessible which includes key-value pairs in storage, balances of accounts and codes of smart contracts.

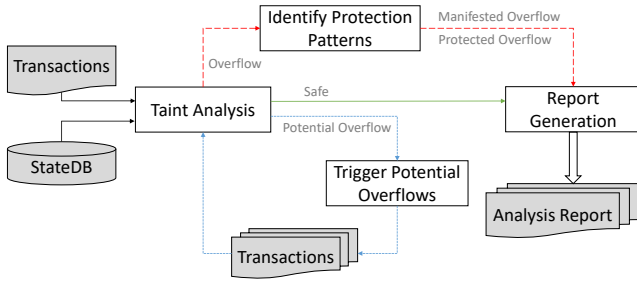


Fig. 2: The high-level framework of EASYFLOW .

Taint Analysis component tracks taints inside the EVM interpreter to monitor the transaction process. The transactions are first analyzed by Taint Analysis component and the smart contracts are divided into three categories: safe, overflow and potential overflow. Report Generation will output analysis reports for safe contracts directly. EASYFLOW will try to identify protection patterns in overflows, and refine them into manifested overflows and protected overflows. When potential overflows are inferred, EASYFLOW will generate a series of transactions in order to trigger the potential overflows via re-execution. As long as any one of the generated transactions is manifested overflow, it means the original potential overflow has been triggered.

In this way, all the smart contracts in transactions can be classified into five categories: safe, manifested overflow, protected overflow, potential overflow triggered and potential overflow not triggered.

#### A. Taint Analysis at EVM Level

EASYFLOW extends the EVM interpreter by tracking the propagation of taints during transactions execution. Taking the batchTransfer function in Fig. 1 as an example again, the taints are introduced when the parameters `_receivers` and `_value` are loaded. When EASYFLOW executes instructions corresponding to the source code in line 4, the values and taint marks of `cnt` and `_value` are moved to the top of stack and `MUL` is then executed to calculate the value of `amount`. As `MUL` is a susceptible integer arithmetic instruction, EASYFLOW will check both the marks and values involved in this instruction. This instruction is safe if the two multipliers are not taints and the protection patterns will be further analyzed when the multiplication result overflows. If more than one multipliers is tainted but the multiplication result does not overflow, EASYFLOW infers that it is a potential overflow and try to trigger it later.

#### B. Identify Protection Patterns

Experienced developers have proposed some schemes to protect their smart contracts from overflow attacks, leading to high False Positives in detection tools. The most commonly used library for protection is named SafeMath<sup>1</sup> which provides several functions for developers to replace ordinary arithmetic operations in contracts. The protection scheme behind

SafeMath is using `require` or `assert` function to check the result after calculation and the contract will throw when an overflow occurs. Some variant libraries and protection codes are also widely adopted and have similar logics and patterns to SafeMath. Therefore, it is not difficult to enumerate all the patterns of protection schemes.

In addition, to our best knowledge, there are also some instructions never protected by existing schemes including `EXP`, `ADDMOD` and `MULMOD`, and some developers do not use SafeMath or other protection schemes properly.

#### C. Trigger Potential Overflows

An Ethereum transaction takes input data and message value as external data and all the internal data is stored in StateDB. Input data is the payload that the sender sends in a transaction to call a specific function in a smart contract and passes in parameters, and message value indicates how many ETHs are sent. The format of input data is a 32-bit function signature and several 256-bit integers, each of which represents a parameter, the position of an array or the length of an array.

To trigger an inferred potential overflow, We implement a straight-forward algorithm in EASYFLOW . The first generated transaction has the same input data as the original transaction but `MAX_UINT256` as message value. Then the 256-bit integers in input data are split and each is assigned to 0 and `MAX_UINT256`. EASYFLOW takes all the possible combinations of these integers as input data and the original message value to generate transactions for re-execution.

### III. USING EASYFLOW

As Fig. 3 illustrates, EASYFLOW can be divided into four parts in implementation, extended go-ethereum, log analyzer, transaction constructor and report generator. Ethereum runtime bytecode and transaction input data can be passed into EASYFLOW , and Solidity source code of smart contracts will be compiled using a built-in official solc in advance. The default StateDB is empty, and state information can be passed in via a JSON-format file before detection, or a running Ethereum node can be connected via remote procedure call (RPC) to provide the real-time StateDB.

**Extended go-ethereum.** Extended go-ethereum was developed based on official golang implementation of the Ethereum protocol, aiming at tracing the propagation of taint data. Tainted stack and tainted memory followed Ethereum stack and memory in implementation, but recoding taint marks instead of specific values while EVM running. Taint detector can mark input data as a taint via inspecting instructions and analyzing their intents, e.g., `CALLDATALOAD` can usually introduce taints into stack, however, when it appears in the front of bytecode and followed by `DIV`, `PUSH FFFFFFFF`, `AND` and `PUSH XXXXXXXX, EQ`, it is used to select proper function by comparing the first 32 bits of input data with function signatures. Overflow detector will check both taint marks and values when EVM executes susceptible integer arithmetic instructions, such as `ADD`, `SUB`, `MUL` and `EXP`. To reduce False Positive, in case of smart contract developers using SafeMath

<sup>1</sup>[https://openzeppelin.org/api/docs/math\\_SafeMath.html](https://openzeppelin.org/api/docs/math_SafeMath.html)

library or conditional statements to protect contracts from overflow, pattern recognizer will try to confirm whether it is a manifested overflow or a protected overflow by matching bytecode to protection patterns. Extended go-ethereum will output trace logs in JSON format as soon as finishing dynamic bytecode execution.

**Log Analyzer.** Log Analyzer analyzes logs generated by extended go-ethereum, and distributes tasks to corresponding components. Log parser first extracts the overflow detection result from logs. Safe, manifested overflow, protected overflow and re-executed potential overflow transactions will be sent to report generator. Transaction constructor will receive potential overflow transactions to conduct further analysis.

**Transaction Constructor.** Transaction Constructor is implemented for re-executing potential overflow transactions with constructed input data. As indicated in II-C, Transaction Constructor splits input data, combines new values of input data and message value, and constructs new transactions. After construction, the transactions will be re-executed by extended go-ethereum to discover whether the potential overflow vulnerability can be triggered.

**Report Generator.** Report generator gathers every received result, and extracts key information into a brief analysis report. All the log files are also attached, and can be accessed through links in report.

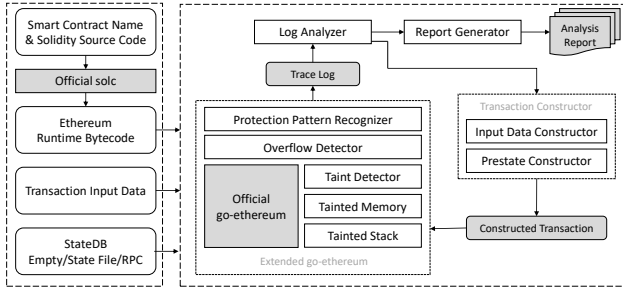


Fig. 3: The architecture of EASYFLOW in implementation

We have deployed EASYFLOW as a web service<sup>2</sup>, and a command-line tool having the same functions can be used offline on Linux as well. Fig. 4 shows a screenshot of the web page of EASYFLOW. EASYFLOW provided ten smart contract examples containing contract name, Solidity source code, runtime bytecode and input data. By clicking the button below, EASYFLOW will automatically analyze overflow vulnerabilities of the selected smart contract, and generate a succinct analysis report at the bottom. Users can also modify contents of the examples or type brand new contract code and input data into the web page, and get the specific analysis report with "one-click".

<sup>2</sup><http://easyflow.cc/>

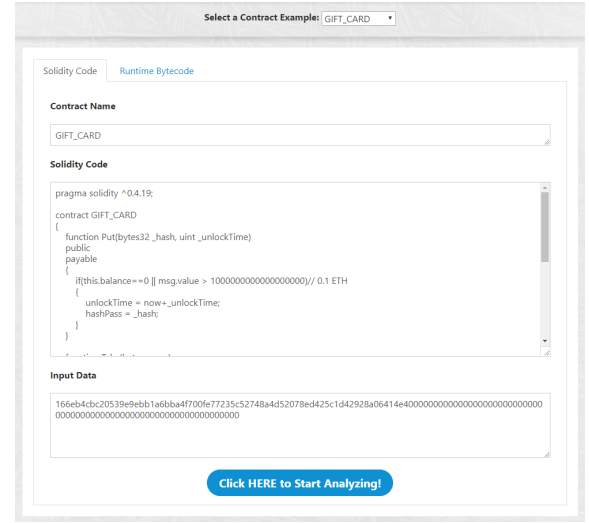


Fig. 4: EASYFLOW Screenshot

Taking the smart contract named GIFT\_CARD as an example, which is partially shown in Fig. 4, we briefly describe the structure of the report in Fig. 5. The report consists of analysis result and analysis log. A sentence of conclusion is used as the analysis result, and input data and result of all the executed transactions are included in the analysis report. Users can also download full trace logs of each transaction via the link on the right of the transaction number. All the log files are in JSON format and can be easily read by machine for further processing.

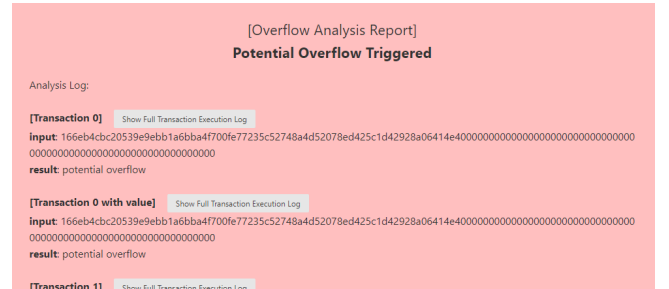


Fig. 5: EASYFLOW Analysis Report

#### IV. PRELIMINARY EVALUATION

We have preliminarily evaluated EASYFLOW on Ethereum Mainnet and the detection results in TABLE I show that EASYFLOW is effective in detecting overflow vulnerabilities in smart contracts and can successfully identify protection patterns and trigger potential overflows.

TABLE I: Detection results

Detection Result	Number of Tx
Manifested Overflow	465
Protected Overflow	6
Potential Overflow Triggered	3871
Potential Overflow Not Triggered	42580
Safe	19914

All the experiments were conducted on Amazon Web Services Cloud (AWS). The virtual machine that deploys EASYFLOW has 1 vCPU, 512MB RAM, and Ubuntu Server 16.04 LTS - Xenial (HVM) as the OS. An Ethereum full node provides StateDB via RPC and it is running on a virtual machine having 4 vCPUs, 32GB RAM, SSD, and the same OS. For most of the smart contracts, EASYFLOW finished analysis in 300ms in command-line mode, and the execution time increases linearly compared to official go-ethereum.

TABLE II: Analysis results on example smart contracts. MO: manifested overflow. PO: protected overflow. POT: potential overflow but triggered. PONT: potential overflow not triggered. S: safe.

Contract	MO	PO	POT	PONT	S	Overhead
BecToken	✓					104.35%
SMT	✓					98.15%
Lizun		✓				100.00%
darx		✓				103.45%
Gift_Card			✓			496.15%
RedEnvelope			✓			294.34%
SimpleLotto				✓		750.00%
HBTOKEN				✓		631.03%
Rating					✓	92.31%
Danksignals					✓	110.71%

The analysis results of some example smart contracts are shown in TABLE II and the addresses of the smart contracts and transactions can be accessed on Github<sup>3</sup>. BecToken is the most famous contract having an overflow vulnerability, and was exploited because of unprotected multiplication. SMT has the similar vulnerability because of unprotected addition. Lizun and darx are protected from overflow vulnerability using SafeMath Library and assertions. GIFT\_CARD is a simple addition overflow, and EASYFLOW successfully triggered the vulnerability by automatically constructing input data based on the real-world transaction. The triggered overflow vulnerability of RedEnvelope is caused by message value. It cannot be exploited in the real world at this time as the amount of ETH is limited, but may be exploitable with the increase of ETH. SimpleLotto actually has an overflow vulnerability, but the contract had committed suicide and all the state of this contract was cleared so that the vulnerability cannot be triggered without manually constructing state information. The potential subtraction overflow in HBTOKEN is protected by conditional statements, and can never be triggered by any input data. There are not susceptible instructions in the transactions of Rating and DANKSIGNALS, and they are both considered to be safe.

## V. RELATED WORK

Research on integer overflow has been in progress for decades and many effective detection schemes were proposed. RICH [3], BRICK [4], SmartFuzz [5] and AIR [6] were developed to detect integer overflows. Dietz *et al.* [7] focused on the integer overflows in C and C++ code. Sun *et al.* [8] showed that not all the integer overflows were malicious and it

could be checked using equivalence checking across multiple precisions.

As blockchain becomes popular in both academia and industry, the smart contract bug detection are noted by researchers. Oyente [9] is a symbolic execution tool built to find potential security bugs of Ethereum contracts. ZEUS [10] presents a formal verification framework that can build and verify correctness and fairness policies. Some other schemes and tools [11] [12] are also presented to analyze smart contracts for vulnerabilities. In contrast, EASYFLOW focus more on integer overflow vulnerabilities, and it is capable of detecting various types of overflows in real-world contracts via dynamic taint analysis, pattern matching and transaction construction. Moreover, EASYFLOW is more lightweight and fast, and can be easily used through browser.

## VI. CONCLUSION

In this demo paper, we present EASYFLOW, a virtual machine level detector for overflow vulnerabilities in Ethereum. The key insight behind EASYFLOW is a taint analysis based tracking technique to monitor real transactions. Particularly, EASYFLOW captures manifested overflows, flags well-protected overflows, infers and triggers potential overflows as well. We managed to leverage EASYFLOW to find real overflows in deployed smart contracts. In the future, we plan to generalize the technique to diverse settings and applications.

## REFERENCES

- [1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.
- [2] V. Buterin *et al.*, "Ethereum white paper," *GitHub repository*, 2013.
- [3] D. Brumley, T.-c. Chiueh, R. Johnson, H. Lin, and D. Song, "Rich: Automatically protecting against integer-based vulnerabilities," *Department of Electrical and Computing Engineering*, p. 28, 2007.
- [4] P. Chen, Y. Wang, Z. Xin, B. Mao, and L. Xie, "Brick: A binary tool for run-time detecting and locating integer-based vulnerability," in *Availability, Reliability and Security, 2009. ARES'09. International Conference on*. IEEE, 2009, pp. 208–215.
- [5] D. Molnar, X. C. Li, and D. Wagner, "Dynamic test generation to find integer bugs in x86 binary linux programs," in *USENIX Security Symposium*, vol. 9, 2009, pp. 67–82.
- [6] R. B. Dannenberg, W. Dormann, D. Keaton, R. C. Seacord, D. Svoboda, A. Volkovitsky, T. Wilson, and T. Plum, "As-if infinitely ranged integer model," in *Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on*. IEEE, 2010, pp. 91–100.
- [7] W. Dietz, P. Li, J. Regehr, and V. Adve, "Understanding integer overflow in c/c++," in *Proceedings of the 34th International Conference on Software Engineering*. IEEE Press, 2012, pp. 760–770.
- [8] H. Sun, X. Zhang, Y. Zheng, and Q. Zeng, "Inteq: recognizing benign integer overflows via equivalence checking across multiple precisions," in *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on*. IEEE, 2016, pp. 1051–1062.
- [9] 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*. ACM, 2016, pp. 254–269.
- [10] S. Kalra, S. Goel, M. Dhawan, and S. Sharma, "Zeus: Analyzing safety of smart contracts," *NDSS*, 2018.
- [11] 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 Proceedings*. ACM, 2018, pp. 65–68.
- [12] H. Liu, C. Liu, W. Zhao, Y. Jiang, and J. Sun, "S-gram: towards semantic-aware security auditing for ethereum smart contracts," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ACM, 2018, pp. 814–819.

<sup>3</sup><https://github.com/Jianbo-Gao/EasyFlow/tree/master/taint-realworld>