BLOCKEYE: CHASING DEFI ATTACKS ON BLOCKCHAIN

Abstract

Decentralised finance, or DeFi, has recently overtaken other types of applications as the most popular on several public blockchains (like Ethereum). In contrast to conventional finance, DeFi enables users to take part in a range of blockchain financial services (such as collateralizing, borrowing, lending, trading, etc.) using smart contracts at a minimal trust cost. On the other side, DeFi's open nature always presents a wide attack surface, seriously threatening the participants' security cash. In the present study, we suggested BLOCKEYE, a real-time threat identification framework for DeFi initiatives on Ethereum blockchain. The two main skills of BLOCKEYE are: (1) An automatic security study approach that employs symbolic reasoning on data flow of critical service states, such as asset price, to assess if they may be artificially altered from the outside, and identifies potentially susceptible DeFi projects. (2) A susceptible DeFi project is then given an off-chain transaction monitor to deploy. For further security analysis, transactions submitted to not just that project but other projects that are connected to it are gathered. When a crucial invariant specified in BLOCKEYE is violated, such as when the advantage is realized quickly and outweighs the cost, a possible attack is highlighted. We employed BLOCKEYE in a number of well-known DeFi projects and identified previously undisclosed security risks. **A** BLOCKEYE video is available at https://youtu.be/7DjsWBLdlQU.

Keywords: Attack Monitoring, Oracle Analysis, Defi.

Authors

S. Suganthi
Research Scholar
Department of Computer Science
VISTAS
Chennai, India.
dinesh.suganthi@gmail.com

Dr. T. Sree Kala
Associate Professor
Department of Computer Science
VISTAS
Chennai, India.

I. INTRODUCTION

Decentralized finance applications, or DeFi applications, have grown rapidly in recent years in the public blockchain ecosystem, such as Ethereum [1]. Contrary to traditional finance, DeFi apps use a decentralized network (such as blockchain) to benefit from its transparency and openness in order to offer a variety of financial services, like trading, collateralizing, lending, borrowing, etc., all without the need for middlemen.

While the popularity and liquidity of DeFi have grown steadily, its openness also creates a lot of space for outside assaults, which might put the safety of DeFi members' funds in danger. Consider a real-world assault on the bZx project, a DeFi scheme for lending and borrowing (see Figure 1). In this instance, the attacker manipulated crypto asset exchange prices via bZx's oracle depending on other DeFi projects (Kyber and Uniswap), generating a profit from a single atomic transaction.

Figure 1 shows a set of six internal transactions that the attacker carried out, including borrowing (such as transactions 1 and 5), trading (like transactions 2, 3, and 4), and repaying (such as transaction 6) crypto assets (i.e., ETH and sUSD). Then, in the precise sequence depicted in Figure 1, such transactions are combined into a single external transaction that Ethereum executes atomically. The attacker initially borrowed 7,500 ETH obtained from bZx (transaction 1) to fund the attack, after which they traded 4, 417.86 of their borrowed ETH for sUSD with other DeFi projects. Because bZx depends on Kyber and Uniswap as its price feed oracles, both of which are susceptible to large-scale attacks, the attacker has the ability to manipulate the ETH/sUSD exchange rate in bZx to his or her advantage. This was followed by transaction 5, which included borrowing 6, 799.27 ETH while holding 1, 099, 841.39 sUSD, and transaction 6, which involved paying back the 7, 500 ETH that was first borrowed. Thus, the attacker's net gain from transactions 1-6 is 2, 381.41 ETH (after deducting a minor sum of ETH for the gas fee[1]), or $600K.

We emphasize that the key to this form of arbitrage—profiting from buying as well as selling items at several prices—is for an attacker to successfully influence exchange rates of crypto asset pairings, like ETH/sUSD in Fig. 1, by taking advantage of bZx's data dependencies on Kyber and Uniswap.

The DeFi project's security has received relatively less investigation compared to the smart security contracts, which was the topic of numerous earlier study works and tools [2, 3], [4, 5]. In general, existing techniques for identifying low-level defects in smart contracts lack both the commercial structure of a DeFi project and the market in which it operates, making it necessary to fully comprehend both in order to detect these assaults. Below, we have outlined the difficulties in dealing with security-related concerns in DeFi initiatives.

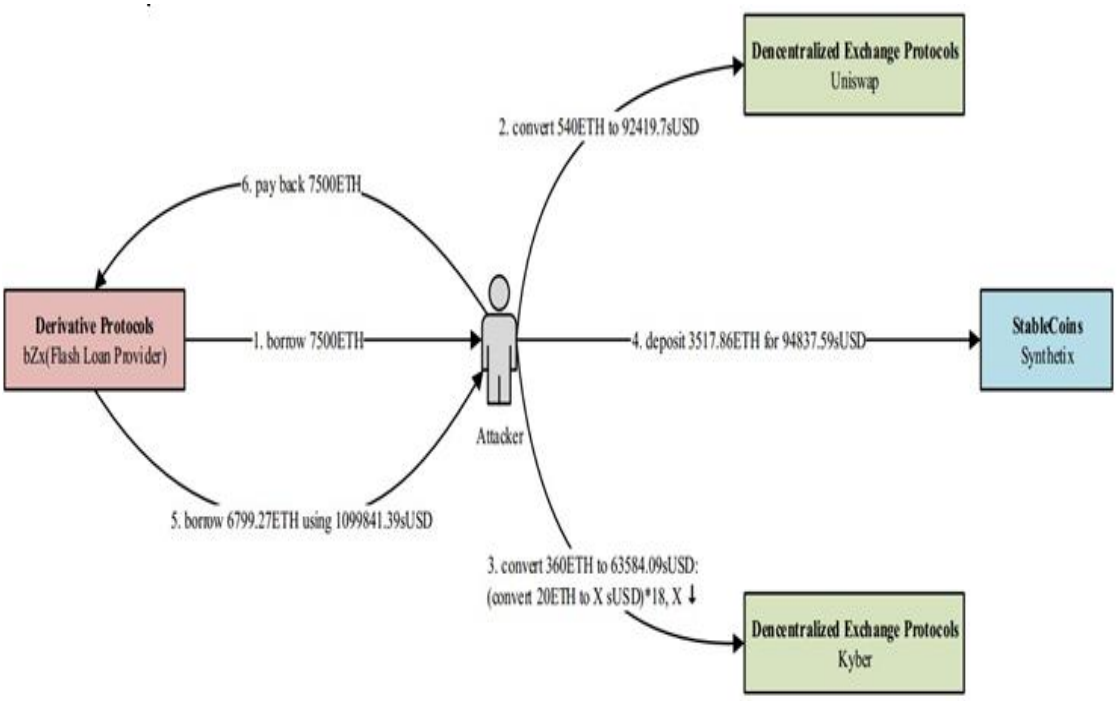

Figure 1: An Attack on the Bzx Project

Challenge 1: Model DeFi Dependency: Model DeFi Dependency is the first challenge. Attacks against DeFi usually consist of more than one project. Since information flow between two DeFi projects is a significant dependency among DeFi projects, its accurate modelling is necessary for identifying such attacks. An abstract analysis can overlook significant high-level business semantics whereas a complete analysis might add excessive complexity.

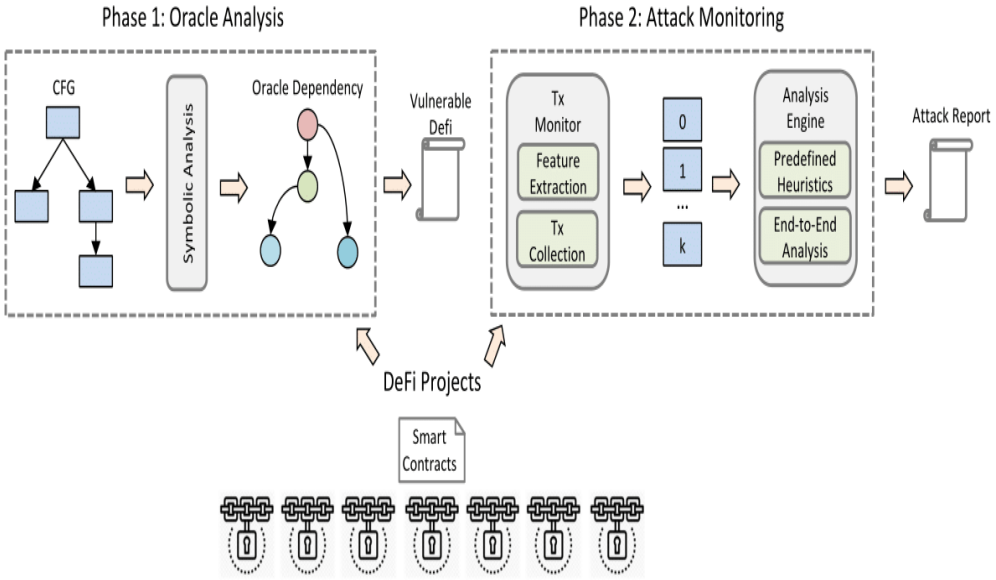

Figure 2: The General BLOCKEYE Workflow

Challenge 2: Understand End-To-End Transactions: Understanding End-To-End Transactions is the second challenge. Additionally, end-to-end analysis, which contrasts the advantages and costs of the transaction sequence, plays a significant role in determining if a sequence of transactions is thought to be malicious. However, utilizing the current infrastructure for blockchain research, such insights are challenging to build and provide.

The BLOCKIE Method. We developed and created BLOCKEYE, the 1st automated threat identification technology for blockchain DeFi initiatives, to address the aforementioned issues. Behind BLOCKEYE are two important realizations. In order to rationalize significant data flow (such as asset price) among related DeFi projects, BLOCKEYE first does a symbolic analysis. This method finds initiatives that could be at risk. Then, in order to quickly identify possible attacks on exposed DeFi projects, BLOCKEYE instals a runtime monitor. In particular, an "end-to-end economic" study is carried out to note fraudulent transactions on the basis of predetermined heuristics, such as excessive gains earned quickly. Then, using BLOCKEYE, we identified possible threats against a number of well-known Ethereum DeFi initiatives.

II. DETECTION OF ATTACKS FOR DEFI

1. **Overview:** Figure 2 displays BLOCKEYE's typical procedure. BLOCKEYE specifically operates in two phases. In 1st step, BLOCKEYE conducts a symbolic analysis of smart contracts from a particular DeFi project. In order to do this, our team extended SERAPH [6, an underlying smart contract analyzer]. The purpose of present stage is to represent the inter-DeFi oracle dependence, or how the services offered by one DeFi are impacted by the oracle data given by another. We mark the DeFi as possibly susceptible when we see Oracle-dependent state modifications. In order to identify external assaults, BLOCKEYE instals a runtime monitor for susceptible DeFi projects during the second phase.

   In order to gather similar transactions based on extracted attributes, such as address, BLOCKEYE, in particular, uses a transaction observation. Then, "end-to-end transactions" are examined using pre-established heuristics, such as turning a sizable profit quickly. When an unusual pattern of transactions is discovered, BLOCKEYE looks for potential threats. Additionally, BLOCKEYE creates a survey report to aid blockchain service suppliers in identifying the issues they have found.

2. **Analysis in Oracle:** As mentioned earlier, BLOCKEYE does oracle analysis to ascertain whether a DeFi is based on an oracle offered by other DeFi. We concentrated particularly on the asset price feed exchanged via oracles.


```
function calculate ContinuousMintReturn(uint _amount)
  public view returns (uint mintAmount) {
  return CURVE.calculatePurchaseReturn(totalSupply(),reserveBalance, uint32(reserveRatio), _amount);
                                                }
function sell(uint  _amount, uint _min) externalreturns (uint _bought) {
  _bought = _sell(_amount);
  require(_bought >= _min, "slippage");
  _burn(msg.sender, _amount); DAI.transfer(msg.sender, _bought);
  ...
```

Figure 3: The EMN Project uses Oracle

III. BLOCKEYE DESIGN

1. **Architecture:** As can be seen in Figure 4, BLOCKEYE is developed as a "web platform with front- and back-end services". The five functional elements that make up this design. At the base, BLOCKEYE adds Oracle analysis capabilities to the smart contract analyzer that was provided before. In this section, Z3 [7] is used as SMT solver.

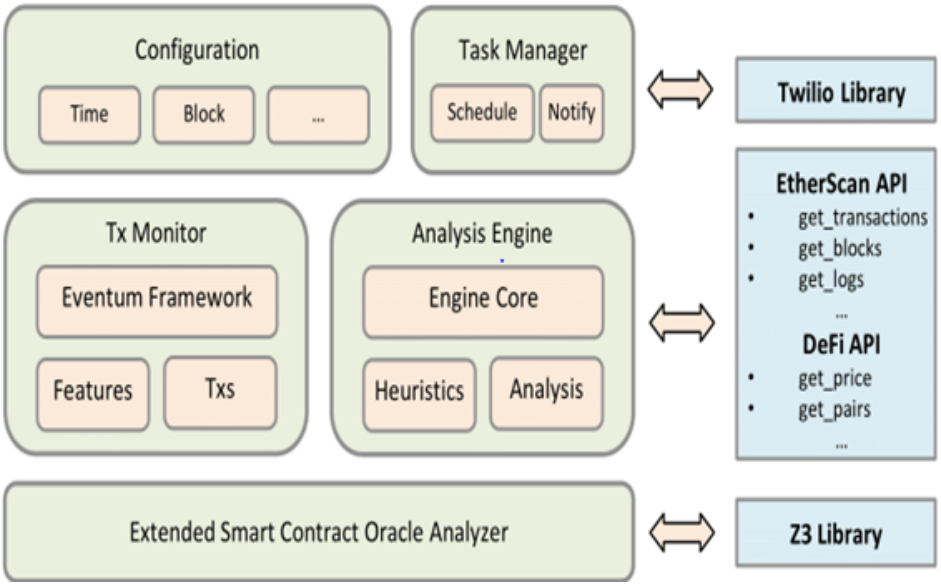

Figure 4: Blockeye General Architecture

2. **Main Performances:** The I/P and O/P BLOCKEYE interfaces are now described, with screenshots revealed in Figures 5 and 6.

Figure 5: The Blockeye Input Interface

According to Figure 5, BLOCKEYE anticipates the input to be DeFi smart contract source code. Both entering the URL of a deployed DeFi project and typing code into the code editor are options available to users. Then, BLOCKEYE will attempt to load the required source code with “Etherscan's source code” retrieval API. The START button may be clicked by users to begin security research on the chosen DeFi project as soon as the smart contract code is made accessible.

An instance of BLOCKEYE output is shown in Figure 6. The findings are broken down into 2 sections: “Oracle Analysis, which displays possible Oracle dependencies observed in the DeFi source code, as well as Attack Monitoring, which gives details on actual attack transactions that transgress the heuristic invariants mentioned in Section II-C. For instance, BLOCKEYE found a dependence between an oracle contract and four smart contract functions in Figure 6, where the oracle contract is defined on line 154 of the code and is triggered by the function compute Continuous BurnReturn in line 168. The relevant state access operation, which is an inquiry to transfer DAI with a dependent quantity of value”, is found on line 242. Additionally, BLOCKEYE indicates a list of the most recent suspicious transactions in Figure 6, every with extensive data. Intrapersonal process. Finally, BLOCKEYE shows a graph of the top attackers and the total amount of attack transactions, which might help users with their future research.

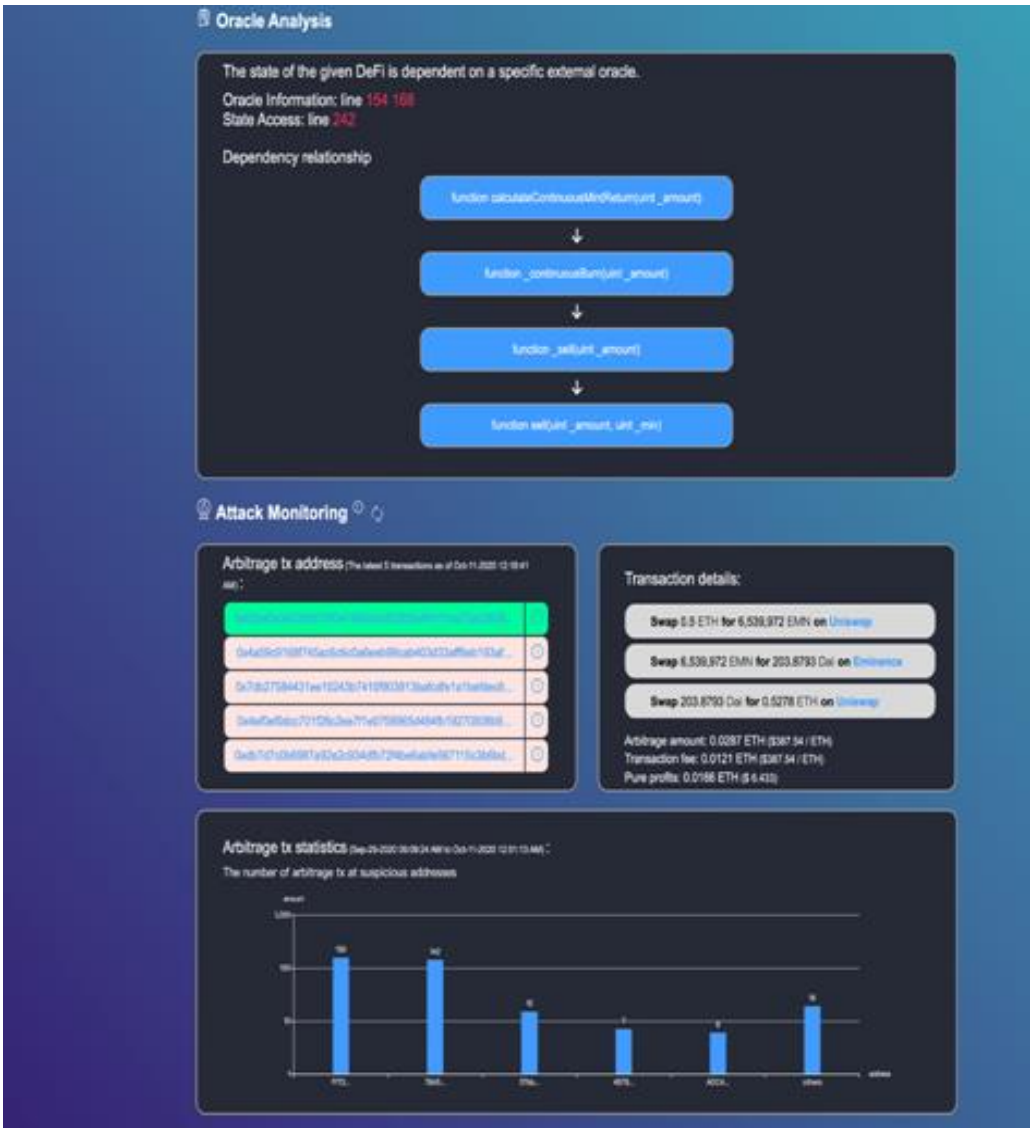

Figure 6: The Blockeye Output Interface

IV. PRELIMINARY EVALUATION

We first assessed BLOCKEYE to confirm its efficiency in identifying oracle-dependent state modifications. The following eight Ethereum DeFi projects were especially taken into account: Aave, DDEX, dYdX, bZx, Compound, Oasis, Nuoand Eminence.
We contrast BLOCKEYE with "Codefi Inspect" [8] in Table I for the purpose of identifying Oracle-dependent state modifications. The findings demonstrate that BLOCKEYE determines each susceptible DeFi in turn, without issuing any false negative or positive alarms. Contrarily, Codefi Inspect improperly disregards DDEX defects, producing a FN (False Negative) outcome.

Table 1: "A Comparison of Blockeye and Codefi Inspect in the Oracle-Dependent State -Update Detection. TP: True Positive; TN: True Negative; FN: False Negative; N/A: Not Available".

"Defi	Codefi Inspect	Blockeye
bZx	TP	TP
DDEX	FN	TP
Aave	TN	TN
dYdX	TN	TN
Compound	TN	TN
Nuo	N/A	TN
Oasis	N/A	TN
Eminence	N/A	TP"

Table 1: With actual transactions on the Ethereum mainnet, we further assessed BLOCKKEYE.

V. RELATED WORK

Recent years have seen a significant increase in media coverage of smart contract security vulnerabilities [4], [5], [3], [2], [6]. For smart contracts, Luu et al. identified four categories of vulnerabilities [2]. Ethereum smart contracts are transformed into datalog logics [9] using a verification approach Tsankov et al. presented [3]. Inductive verification of smart contracts has been also addressed by Permenev et al. [10]. Liu et al. suggested a statistical method to find probable code odours in addition to safety concerns [5]. Previous work has paid relatively little consideration to the security of DeFi initiatives. A number of mathematical and economic models were presented [11], [12], [13], and [14] to conceptually comprehend the risks of DeFi.

VI. CONCLUSION

We have emphasized "BLOCKEYE" as an open platform for identifying DeFi assaults on the blockchain in this paper. In comparison to other smart contract analyzers, BLOCKEYE provides crucial capabilities to model relationships across DeFi projects and quickly identify possible end-to-end assaults. The fundamental concepts of BLOCKEYE are pattern-based runtime transaction validation and symbolic oracle research. With the help of BLOCKEYE, we were able to identify possible threats that had not yet been made public in numerous well-known DeFi initiatives on Ethereum.

REFERENCES

[1] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," Ethereum Project Yellow Paper, vol. 151, 2014.
[2] 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.
[3] P. Tsankov, A. Dan, D. D. Cohen, A. Gervais, F. Buenzli, and M. Vechev, "Securify: Practical security analysis of smart contracts," arXiv preprint arXiv:1806.01143, 2018.
[4] C. Liu, H. Liu, Z. Cao, Z. Chen, B. Chen, and B. Roscoe, "Reguard: finding reentrancy bugs in smart contracts," in ICSE (Companion). ACM, 2018, pp. 65–68.

[5] H. Liu, C. Liu, W. Zhao, Y. Jiang, and J. Sun, "S-gram: towards semantic-aware security auditing for ethereum smart contracts," in ASE. ACM, 2018, pp. 814–819.
[6] Z. Yang, H. Liu, Y. Li, H. Zheng, L. Wang, and B. Chen, "Seraph: enabling cross-platform security analysis for evm and wasm smart contracts," in Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings, 2020, pp. 21–24.
[7] "Microsoft z3 smt solver," https://z3.codeplex.com/, 2019.
[8] "Codefi inspect," https://inspect.codefi.network/, 2020.
[9] T. Eiter, G. Gottlob, and H. Mannila, "Disjunctive datalog," ACM Transactions on Database Systems (TODS), vol. 22, no. 3, pp. 364–418, 1997.
[10] A. Permenev, D. Dimitrov, P. Tsankov, D. Drachsler-Cohen, and M. Vechev, "Verx: Safety verification of smart contracts," Security and Privacy, vol. 2020, 2019.
[11] K. Qin, L. Zhou, B. Livshits, and A. Gervais, "Attacking the defi ecosystem with flash loans for fun and profit," arXiv preprint arXiv:2003.03810, 2020.
[12] J. Kamps and B. Kleinberg, "To the moon: defining and detecting cryptocurrency pump-and-dumps," Crime Science, vol. 7, no. 1, p. 18, 2018.
[13] B. Liu and P. Szalachowski, "A first look into defi oracles," arXiv preprint arXiv:2005.04377, 2020.
[14] L. Gudgeon, D. Perez, D. Harz, A. Gervais, and B. Livshits, "The decentralized financial crisis: Attacking defi," arXiv preprint arXiv:2002.08099, 2020.

