CLUE: Towards Discovering Locked Cryptocurrencies in Ethereum

Xiaoqi Li
Department of Computing, The Hong Kong Polytechnic University
Hong Kong SAR, China
csxqli@gmail.com

Xiapu Luo
Department of Computing, The Hong Kong Polytechnic University
Hong Kong SAR, China
csxluo@comp.polyu.edu.hk

Ting Chen
Center for Cybersecurity, University of Electronic Science and Technology of China
Chengdu, China
brokendragon@uestc.edu.cn

Chenxu Wang
School of Software Engineering, Xi’an Jiaotong University
Xi’an, China
cxwang@mail.xjtu.edu.cn

1 INTRODUCTION

As the most popular blockchain that supports smart contracts, there are already more than 296 thousand kinds of cryptocurrencies built on Ethereum. However, not all cryptocurrencies can be controlled by users. For example, some money is permanently locked in wallets’ accounts due to attacks. In this paper, we conduct the first systematic investigation on locked cryptocurrencies in Ethereum. In particular, we define three categories of accounts with locked cryptocurrencies and develop a novel tool named Clue to discover them. Results show that there are more than 216 million dollars value of cryptocurrencies locked in Ethereum. We also analyze the reasons (i.e., attacks/behaviors) why cryptocurrencies are locked. Because the locked cryptocurrencies can never be controlled by users, avoid interacting with the accounts discovered by Clue and repeating the same mistakes again can help users to save money.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SAC’21, March, 2021, Guangju, Korea
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06... $15.00
https://doi.org/10.1145/nnnnnn.nnnnnn

ABSTRACT

As the most popular blockchain that supports smart contracts, there are already more than 296 thousand kinds of cryptocurrencies built on Ethereum. However, not all cryptocurrencies can be controlled by users. For example, some money is permanently locked in wallets’ accounts due to attacks. In this paper, we conduct the first systematic investigation on locked cryptocurrencies in Ethereum. In particular, we define three categories of accounts with locked cryptocurrencies and develop a novel tool named Clue to discover them. Results show that there are more than 216 million dollars value of cryptocurrencies locked in Ethereum. We also analyze the reasons (i.e., attacks/behaviors) why cryptocurrencies are locked. Because the locked cryptocurrencies can never be controlled by users, avoid interacting with the accounts discovered by Clue and repeating the same mistakes again can help users to save money.

CCS CONCEPTS

• Security and privacy → Distributed systems security;
• Software security engineering; • Software and its engineering → Software usability.

KEYWORDS

Distributed System Security, Ethereum, Cryptocurrency

ACM Reference Format:
Xiaoqi Li, Ting Chen, Xiapu Luo, and Chenxu Wang. 2021. CLUE: Towards Discovering Locked Cryptocurrencies in Ethereum. In Proceedings of the 36th ACM/SIGAPP Symposium on Applied Computing (SAC’21). ACM, New York, NY, USA, Article 4, 4 pages. https://doi.org/10.1145/nnnnnn.nnnnnn
2 BACKGROUND AND RELATED WORK

Ethereum: Ethereum is the most popular blockchain system that supports smart contracts [5]. There are two kinds of accounts in Ethereum, i.e., EOA and contract account [6]. EOA is controlled by user through its private key, which does not store any code. Contract account is created by EOA or another contract, which stores the runtime bytecodes of the contract. Smart contract is a program deployed and executed in blockchain [7]. Every node in Ethereum runs an EVM (Ethereum Virtual Machine) and the runtime bytecodes are executed in the EVM. When a user calls the smart contract, he/she needs to send transaction with gas to the address of the target contract [8]. Every operation of contract runtime bytecodes consumes specific amount of gas when they are executed in the EVM [9]. Developers/users can also destruct the deployed smart contract through executing SELFDESTRUCT operation [10].

Cryptocurrency: Cryptocurrency is digital assets based on blockchain techniques [11]. There are two categories of cryptocurrencies in Ethereum, i.e., ETH and CBC (Contract-Based Cryptocurrency) [11]. ETH is the native cryptocurrency of Ethereum. Apart from ETH, there are many other kinds of cryptocurrencies based on contracts, and ERC20 is the most popular standard of CBC [6]. All the CBC analyzed in this paper are compliant with ERC20. Both EOA and contract account can hold cryptocurrency. EOA can transfer out ETH by initiating transactions from it, and contract can transfer out ETH by executing specific operations (e.g., CALL, SELFDESTRUCT). Note that accounts can only transfer out their CBC by calling the corresponding ERC20-based smart contract. The ERC20 standard provides some basic functions and events that must be implemented of CBC in Ethereum. If the user $U_a$ wants to transfer out CBC, $U_a$ can call transfer(). Furthermore, the user $U_a$ can authorize another account $U_b$ to transfer out CBC through calling transferFrom(). Before $U_b$ transfers out $U_a$’s CBC, $U_a$ must authorize the account $U_b$ through calling approve().

Account State: Every account in Ethereum has four state fields stored in StateDB (State Database) [4]. For each account, we mainly analyze three fields. Code $\sigma[a]_c$ stores the smart contract’s runtime bytecodes, which is empty if $a$ is an EOA. Balance $\sigma[a]_b$ stores the ETH balance value (in Wei) of the account. Nounce $\sigma[a]_n$ stores the number of transactions sent from EOA, or the number of contracts created by contract account.

Related Work: Chen et al. [12] detected Ponzi schemes, which are classic frauds and might cheat users’ ETH. They built a classification model to detect latent Ponzi schemes by using data mining and machine learning methods. Cheng et al. [13] analyzed the attack that steals cryptocurrencies exploiting unprotected JSON-RPC endpoints. They designed and implemented a honeypot that could capture real attacks in the wild. Ji et al. [14] implemented a tool named Deposafe to detect and exploit the fake deposit vulnerability in ERC-20 tokens. However, all of the above work analyzed the cryptocurrencies illegally possessed by criminals, and they did not analyze locked cryptocurrencies that does not belong to anyone. [15] measured the network properties and structures of ERC20 smart contracts, and [11] analyzed inconsistent behaviors in ERC20 smart contracts. However, they focused on analyzing the smart contracts’ implementations and invocations, whose purposes differ from ours. There are some other work analyzed cryptocurrencies in Ethereum [16] or other blockchains [17].

3 LOCKED CRYPTOCURRENCIES

Destructed Contract: In Ethereum, the smart contract can be destructed and transfer out all its stored ETH through executing the SELFDESTRUCT operation. After destruction, the smart contract account will be deleted from StateDB. However, some users may not know in time of the smart contract’s destruction and still send ETH/CBC to it, which leads to the sent ETH/CBC be locked. After sending ETH to the destructed smart contract account, the contract account with the same address before destruction will be created again in the StateDB. For the CBC held by the destructed contract account, most of it will also be permanently locked. Because the destructed contract account stores no runtime bytecodes, it cannot send out transaction. Therefore, the destructed contract account cannot transfer out its CBC through calling transfer() function, or authorize another account to transfer out its CBC through calling the transferFrom() function. Above all, all the ETH and most of the CBC held by the destructed contract accounts are permanently locked in Ethereum. For example, one smart contract named InsightsNetworkContributions (Address: 0x97eC9FbB...) is discovered by Clue as destructed contract with locked cryptocurrencies. It has been transferred more than 208 ETH after its destruction, which is worth more than 79 thousand dollars.

Attacked Parity Contract: In 2017, the attacker escalated his privilege and destructed the multi-sig library of Parity wallets, leading to all the ETH and most of the CBC held by wallet contracts that depend on the library locked permanently [4]. The attacker destructed the wallets’ library in the following process. First, the attacker called the library’s functions initWallet() and initMultiowned() through the fallback function, to escalate his/her privilege. Second, the attacker destructed the library contract through calling function kill(). After the library’s destruction, all the wallet contracts can no longer call the library and executing its functions. Therefore, all the ETH stored in the attacked Parity wallet contracts is permanently locked. Furthermore, all the CBC held by the attacked wallet contracts is also locked. This is because the wallet contract cannot call the ERC20 contracts. For example, one attacked Parity wallet contract (Address: 0x0da3cB30, ...) discovered by Clue stores 2,576.35ETH. After the attack, there
Transaction analysis
Locked cryptocurrencies in
Locked cryptocurrency
with locked cryptocurrencies can be detected and alerted in time,
tailed information of the transaction. Because the execution of
SELFDESTRUCT operation, which is used for destructing the con-
deviation of the execution of the transaction's execution
and detailed information, we analyze the specific address
of the destructed contract account. If the type field of one internal
transaction is "suicide", we can conclude that it is used for destruc-
ting the contract account. Then we export the sender address of the
internal transaction, which is the address of destructed contract.
Fourth, we analyze ETH/CBC balance of the destructed contract
through Ethereum RPC-APIs. At last, the destructed contracts with
locked cryptocurrencies can be discovered.

\textbf{Detection of Destructed Contracts:} The detection of de-
structed Parity contracts with locked cryptocurrencies is divided into
four steps. First, for contract accounts stored in the StateDB, we ex-
port their runtime bytecodes from the code field $\sigma[a]_{\text{c}}$. Second, we
statically analyze the bytecodes. In particular, we use \texttt{DISASM} \cite{18}
to disassemble the runtime bytecodes and detect hardcoded Parity
library pattern. Third, we leverage symbolic execution techniques
to analyze the runtime bytecodes with the hardcode pattern. We
use \texttt{Oyente} \cite{19} as the symbolic execution engine. During the
symbolic execution process of runtime bytecodes, we monitor the
external call related operations. If we encounter external call oper-
ation's execution, we analyze its second operand $P_a$, which is used
for the target address of the external call. If $P_a$ is a real value and
equals with the hardcoded Parity library's address, we can conclude
that the corresponding analyzed contract account is an attacked
Parity contract. Furthermore, the attacked Parity contract cannot
call ERC20 contracts to transfer out its CBC. This is because $P_a$
does not equal with ERC20 contracts’ addresses or associated with
transaction's input data. Fourth, we analyze the ETH/CBC balances
for the detected contracts in the third step.

\textbf{Detection of Contract-creation Failure EOAs:} We leverage
account state analysis and transaction analysis to detect contract-
creation failure EOAs with locked cryptocurrencies, which is di-
vided into three steps. First, we traverse the StateDB and filter out
sensitive EOAs. The sensitive EOAs have the following state fea-
tures: nonce $\sigma[a]_{n}$ is zero, and code $\sigma[a]_{c}$ is empty. The sensitive
EOAs with these features never send out any transaction. $\sigma[a]_{c}$
field is empty indicates that the account $a$ is an EOA. For an EOA,
its $\sigma[a]_{n}$ field stores the number of transactions sent from it. Sec-
ond, leveraging Ethereum RPC-API, we fetch and analyze sensitive
EOA’s oldest transaction, to verify that it encountered an error

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure.png}
\caption{Overview of \textsc{Clue}’s architecture. There are mainly two modules in \textsc{Clue}: detection of locked cryptocurrencies in contract and detection of locked cryptocurrencies in EOA.}
\end{figure}

were 17.88ETH transferred to this wallet contract. Furthermore,
there is also 2.09S value of CBC locked in it. If these wallet accounts
with locked cryptocurrencies can be detected and alerted in time,
the users might no longer transfer cryptocurrencies to it, which
can help users to save money.

\textbf{Contract-creation Failure EOA:} When the user deploys a
smart contract in Ethereum, he/she will still receive one fake con-
tract address if the contract-creation fails. Indeed, the received
contract address does not exist in StateDB just after the contract-
creation failure. However, some users might wrongly ignore the fail-
ure message and still transfer cryptocurrencies to the fake contract
address, leading to cryptocurrencies locked permanently. Because
the address with locked cryptocurrencies never stores code, we
classify it as EOA. For Example, one EOA (Address: 0x5488b0a0...)
discovered by \textsc{Clue} locks 19 ETH in value of 7,088.71 dollars and
some CBC in value about seven dollars. After the user encountered
an out-of-gas error during contract-creation, he still called the fake
contract address three times.

\section{CLUE}

The overview of \textsc{Clue}’s architecture is shown in Figure 1, which
mainly consists of two modules: (1) Locked cryptocurrencies in
contracts. For destructed contract, we debug accounts’ historical
transactions and detect destructed contracts through transaction
trace analysis and balance analysis. For attacked Parity contract, we
statically analyze contracts’ runtime bytecodes and detect wallet
contracts through symbolic execution. (2) Locked cryptocurrencies
in EOAs. We export sensitive EOAs from StateDB and detect
contract-creation failure accounts with locked cryptocurrencies
through transaction and balance analysis.

\textbf{Detection of Destructed Contracts:} The detection of de-
structed contracts with locked cryptocurrencies is divided into
four steps. First, for accounts stored in the StateDB, we de-
bug their historical external transactions through Geth API
\texttt{debug.traceTransaction()}. From the execution trace of the
external transaction, we analyze whether it ever executed the
SELFDESTRUCT operation, which is used for destructing the con-
tact account. Second, for the external transaction that executed
SELFDESTRUCT, we leverage Ethereum RPC API to get the de-
tailed information of the transaction. Because the execution of
SELFDESTRUCT will produce internal transaction, we get the de-
tailed information of the internal transaction according to the hash
of external transaction. Third, leveraging the transaction’s execu-
tion trace and detailed information, we analyze the specific address
of the destructed contract account. If the type field of one internal
transaction is "suicide", we can conclude that it is used for destruc-
ting the contract account. Then we export the sender address of the
internal transaction, which is the address of destructed contract.
Fourth, we analyze ETH/CBC balance of the destructed contract
through Ethereum RPC-APIs. At last, the destructed contracts with
locked cryptocurrencies can be discovered.

\textbf{Detection of Attacked Parity Contracts:} The detection of at-
tacked Parity contracts with locked cryptocurrencies is divided into
four steps. First, for contract accounts stored in the StateDB, we ex-
port their runtime bytecodes from the code field $\sigma[a]_{\text{c}}$. Second, we
statically analyze the bytecodes. In particular, we use \texttt{DISASM} \cite{18}
to disassemble the runtime bytecodes and detect hardcoded Parity
library pattern. Third, we leverage symbolic execution techniques
to analyze the runtime bytecodes with the hardcode pattern. We
use \texttt{Oyente} \cite{19} as the symbolic execution engine. During the
symbolic execution process of runtime bytecodes, we monitor the
external call related operations. If we encounter external call oper-
ation’s execution, we analyze its second operand $P_a$, which is used
for the target address of the external call. If $P_a$ is a real value and
equals with the hardcoded Parity library’s address, we can conclude
that the corresponding analyzed contract account is an attacked
Parity contract. Furthermore, the attacked Parity contract cannot
call ERC20 contracts to transfer out its CBC. This is because $P_a$
does not equal with ERC20 contracts’ addresses or associated with
transaction’s input data. Fourth, we analyze the ETH/CBC balances
for the detected contracts in the third step.

\textbf{Detection of Contract-creation Failure EOAs:} We leverage
account state analysis and transaction analysis to detect contract-
creation failure EOAs with locked cryptocurrencies, which is di-
vided into three steps. First, we traverse the StateDB and filter out
sensitive EOAs. The sensitive EOAs have the following state fea-
tures: nonce $\sigma[a]_{n}$ is zero, and code $\sigma[a]_{c}$ is empty. The sensitive
EOAs with these features never send out any transaction. $\sigma[a]_{c}$
field is empty indicates that the account $a$ is an EOA. For an EOA,
its $\sigma[a]_{n}$ field stores the number of transactions sent from it. Sec-
ond, leveraging Ethereum RPC-API, we fetch and analyze sensitive
EOA’s oldest transaction, to verify that it encountered an error
and returned a smart contract address. As described in Section 3, the contract-creation transaction will also return a fake contract address when it fails with errors. Third, we analyze ETH/CBC balance of the detected EOAs in the second step through Ethereum RPC-APIS. At last, contract-creation failure accounts with locked cryptocurrencies can be discovered.

5 EVALUATION

We carry out experiments to answer the following research questions: RQ1 (Quantity): How much value of locked cryptocurrencies can be detected by CLUE? RQ2 (Accuracy): To what extent can CLUE accurately discover locked cryptocurrencies?

Table 1: Statistics of locked cryptocurrencies and related accounts detected through CLUE. (O: discovered candidate accounts. ◦: accounts with locked cryptocurrencies.)

| Category                        | Discovered account | Locked ETH | Locked CBC |
|---------------------------------|--------------------|------------|------------|
| Attacked Parity contract        | 4763 J. 190,040     | 25,036,305  | 3,720,537  |
| Contract-creation failure EOA   | 5903 J. 190,199      | 25,886,741  | 3,720,537  |
| Destructed contract             | 5902,454 J. 3,720   | 25,886,741  | 3,720,537  |
| Total                           | 5902,454 J. 3,720   | 25,886,741  | 3,720,537  |

RQ1 Quantity: We evaluate the quantity of locked cryptocurrencies detected by CLUE, whose statistics are shown in Table 1. Applying CLUE to all Ethereum StateDB data, we totally discover 216,186,551,125 value of locked cryptocurrencies. The related accounts’ addresses for each category and analyzed transaction data are published on https://figshare.com/articles/dataset/11605296. For the destructed contracts, many of them were created due to DoS attacks [20]. The attacker created large amount of smart contracts and destructed them through SELFDESTRUCT operation. Most of these destructed contracts are not called any more by normal users. Therefore, most of the destructed contracts do not lock any cryptocurrency. For the attacked Parity contracts, we totally discover 658 related accounts, while Etherscan only tags 153 of them. For contract-creation failure EOAs, their locked cryptocurrencies’ value is small, because users might stop calling these accounts after they realize the contract-creation failure. The locked CBC of destructed contracts does not be transferred out during contracts’ destruction, which leads to more locked CBC than ETH. Furthermore, all these detected accounts might lock more cryptocurrencies with Ethereum’s running, and we also plan to measure locked cryptocurrencies’ time accumulation in our future work. **Answer to RQ1:** For the proposed three kinds of Ethereum accounts, we totally discover 216,186,551,125 value of cryptocurrencies locked in them.

RQ2 Accuracy: For destructed contracts, we check all the 173 discovered accounts through Etherscan. All of them have been tagged “Self-Destruct”, and they all have more than zero value of ETH/CBC. Furthermore, there is no ETH/CBC transferred out after their destruction. Similarly, all the 203 attacked Parity wallets have more than zero value of ETH/CBC, and their ETH/CBC never be transferred out after the Parity attack (Transaction hash: 0x47f7f7f7...). In addition, we decompose these contracts leveraging PANORAMIX [21], and they all call the attacked Parity wallets’ library. For the contract-creation failure accounts, we check all the 191 discovered accounts that lock cryptocurrencies through Etherscan. All of these accounts encountered errors during contract-creation, and they all have more than zero value of ETH/CBC. Also, their ETH/CBC is never transferred out. **Answer to RQ2:** 100% of the 567 accounts discovered by CLUE store cryptocurrencies, and all of these cryptocurrencies are locked permanently.

6 DISCUSSION AND CONCLUSION

Discussion: We propose and detect three categories of accounts with locked cryptocurrencies in Ethereum, while there might also exist other categories. In our future work, we plan to analyze more categories of accounts with locked cryptocurrencies. We run CLUE on all the Ethereum accounts’ data. Although the number of discovered accounts with locked cryptocurrencies is small (i.e., 567), the value of locked cryptocurrencies is great. To the best of our knowledge, there is still no research of how many accounts with locked cryptocurrencies exist in Ethereum, and our work fills this gap. In other blockchain systems (e.g., Bitcoin, EOS), there might also exist locked cryptocurrencies. We plan to detect more cryptocurrencies in other blockchain systems in future work.

Conclusion: In this paper, we analyzed cryptocurrencies locked permanently in Ethereum. We defined three categories of accounts with locked cryptocurrencies and implemented a tool named CLUE, which discovered more than 216 million dollars value of locked cryptocurrencies. We also analyzed why these cryptocurrencies are locked, which can help users/developers to avoid losing money.

REFERENCES

[1] "Token tracker." https://etherscan.io/tokens, 2020.
[2] "Total ether supply and market capitalization." https://etherscan.io/stat/supply, 2020.
[3] "Tether usd," https://etherscan.io/token/0xdac17f958d2ee523a2206100546e56e
d3811c7e, 2020.
[4] X. Li, T. Chen, X. Luo, and J. Yu, "Characterizing erasable accounts in ethereum," in Proc. of ISC, 2020.
[5] X. Li, P. Jiang, T. Chen, X. Luo, and Q. Wen, "A survey on the security of blockchain systems," in Future Generation Computer Systems, 2020.
[6] X. Li, T. Chen, X. Luo, T. Zhang, L. Yu, and Z. Xu, "Stan: Towards describing bytedecodes of smart contract," in Proc. of QRS, 2020.
[7] T. Chen, Y. Zhu, Z. Li, J. Chen, X. Li, X. Luo, X. Lin, and X. Zhang, "Understanding ethereum via graph analysis," in Proc. of INFOCOM, 2018.
[8] T. Chen, Z. Li, H. Zhou, J. Chen, X. Luo, X. Li, and X. Zhang, "Towards saving money in using smart contracts," in Proc. of ICSE, 2018.
[9] T. Chen, X. Li, X. Luo, and X. Zhang, "Under-optimized smart contracts devour your money," in Proc. of SANER, 2017.
[10] T. Chen, Y. Feng, Z. Li, H. Zhou, X. Luo, X. Li, X. Xiao, J. Chen, and X. Zhang, "Gaschecker: Scalable analysis for discovering gas-inefficient smart contracts," in IEEE Transactions on Emerging Topics in Computing, 2020.
[11] T. Chen, Y. Zhang, Z. Li, X. Luo, T. Wang, R. Cao, X. Xiao, and X. Zhang, "Token-scope: Automatically detecting inconsistent behaviors of cryptocurrency tokens in ethereum," in Proc. of CCS, 2019.
[12] W. Chen, Z. Zheng, J. Cui, E. Ngi, P. Zheng, and Y. Zhou, "Detecting ponzi schemes on ethereum: Towards healthier blockchain technology," in Proc. of WOIV, 2018.
[13] Z. Cheng, X. Hou, R. Li, Y. Zhou, X. Luo, J. Li, and K. Ren, "Towards a first step to understand the cryptocurrency stealing attack on ethereum," in Proc. of RAID, 2019.
[14] R. Ji, N. He, L. Wu, H. Wang, G. Bai, and Y. Guo, "Deposafe: Demystifying the fake deposit vulnerability in ethereum smart contracts," in arXiv preprint, 2020.
[15] S. Somin, G. Gordon, and Y. Altshuler, "Network analysis of erc20 tokens trading on ethereum blockchain," in Proc. of FCSC, 2018.
[16] M. Frowin, A. Fuchs, and R. Böhme, "Detecting token systems on ethereum," in Proc. of FC, 2019.
[17] S. T. Howell, M. Niessner, and D. Yermack, "Initial coin offerings: Financing growth with cryptocurrency token sales," in The Review of Financial Studies, 2020.
[18] "Thisism," https://github.com/Arcanchid/evmdns, 2019.
[19] L. Luo, D.-H. Chu, H. Olicek, P. Saxena, and A. Hobor, "Making smart contracts smarter," in Proc. of CCS, 2016.
[20] T. Chen, X. Li, Y. Wang, J. Chen, Z. Li, X. Luo, M. H. Au, and X. Zhang, "An adaptive gas cost mechanism for ethereum to defend against under-priced dos attacks," in Proc. of ISPEC, 2017.
[21] "PANORAMIX," https://github.com/eveem-org/panoramix, 2019.