Wallet Contracts on Ethereum

Monika di Angelo, Gernot Slazer
TU Wien, Vienna, Austria
{monika.di.angelo,gernot.salzer}@tuwien.ac.at

***Abstract*—On the blockchain, cryptocurrencies play a role similar to cash, while cryptographic tokens are a universal tool for handling rights and assets. Software wallets interact with blockchains in general and with smart contracts (on-chain programs) in particular. Some wallets are realized (partly) as smart contracts with the intent to increase trust and security by being transparent and by offering features like daily limits, approvals, multiple signatures, and recovery mechanisms.**

Ethereum is the most prominent platform for both, tokens and smart contracts, and thus also for wallet contracts. We discuss several methods for identifying wallet contracts in a semi-automatic manner by looking at the deployed bytecodes and their interaction patterns. Furthermore, we differentiate characteristics of wallets in use, and group them into six types.

***Index Terms*—analysis, EVM bytecode, transaction data**

I. Introduction

Wallets keep valuables, credentials, and items for access rights (like cash, licenses, credit cards, key cards) in one place, for ease of access and use. In this paper, we investigate wallet contracts on the main chain of Ethereum qualitatively as well as quantitatively up to block 8 450 000, mined on August 30, 2019. An extended version of this work can be found at [1].

Methodologically, we start from the source code of wallets and determine characteristic functions. Then we search the deployed bytecode for variants of the wallets with the same profile. Some wallets can also be detected by their creation history or by the way they interact with other contracts. We group the wallets according to their functionality and collect creation and usage statistics from the blockchain data. In total, we found 3.9 M deployed wallet contracts of different 24 types.

This work thus contributes to a better understanding of what smart contracts on Ethereum are actually used for. We extract a comprehensive collection of blueprints for wallet contracts and thereby compile a ground truth. Moreover, the wallet blueprints and the features they implement may serve as a resource when designing further decentralized trading apps.

II. Bytecode Analysis

1) Code Skeletons: To detect functional similarities between contracts we compare their *skeletons*. These are obtained from the bytecodes of contracts by replacing metadata, constructor arguments, and the arguments of PUSH operations uniformly by zeros and by stripping trailing zeros. The rationale is to remove variability that has little impact on the functional behavior, like the swarm hashes added by the Solidity compiler or hard-coded addresses of companion contracts. Skeletons allow us to transfer knowledge gained about one contract to others with the same skeleton. Note that the 18 M contract deployments so far give rise to 252 k distinct bytecodes and just 119 k distinct skeletons. While only 77 k (0.4 %) contracts provide verified source codes on `etherscan.io`, we are able to relate 6.7 M (37 %) of the contracts to corresponding verified source code by exploiting creation histories and the similarity of skeletons.

2) Interface Extraction: We developed a pattern-based tool to extract the interface contained in the bytecode. As ground truth for validation, we used the 77 ϱerified source codes together with the corresponding bytecode and ABI as provided by Etherscan. The signatures extracted by our tool differed from the ground truth in 42 cases. Manual inspection revealed that our tool was correct also in these cases, whereas the ABIs did not faithfully reflect the signatures in the bytecode (e.g. due to compiler optimization or library code).

3) Interface Restoration: To understand the purpose of contracts we try to recover the function headers from the signatures. As the signatures are partial hashes of the headers, we use a dictionary of headers with their 4-byte signatures (collected from various sources), which allows us to obtain a header for 59 % of the 254 k distinct signatures on the main chain. Since signatures occur with varying frequencies and codes are deployed in different numbers, this ratio increases to 89 % when picking a deployed contract at random.

III. Identifying Wallets

We define a wallet as a contract that only manages assets. In contrast, contracts that serve other purposes as well beyond managing assets are termed non-wallet contracts.

Our two-step approach identifies potential wallet contracts and then checks if the bytecode actually implements a wallet. In step 1, we first collect the set of blueprints. We start with a few known wallets that have Solidity sources. As some wallets are deployed in large quantities, we also examine mass deployments (factories) for potential wallets. Finally, we scan all verified source codes for wallets contracts. With this set of blueprints, we fuzz all implemented interfaces in order to capture variants by identifying idiosyncratic signatures of the blueprints and collecting all bytecodes that implement these signatures. In step 2, we check the other signatures and occasionally the bytecode to exclude any non-wallets.

IV. Classification and Comparison of Wallets

Employing the technique of skeletons in combination with the fuzzed interface method, we eventually had to examine 631 distinct code skeletons, corresponding to 1357 distinct

bytecodes deployed at 3.9 M addresses. The skeletons can be grouped into 24 blueprints for wallets. Based on the features a wallet provides, we assign the blueprints to one of six types. Table I lists the blueprints with their names, types, and numbers of deployed instances. A detailed description of each blueprints and its characteristics can be found in [1].

TABLE I
FUNCTIONAL TYPES OF WALLET CONTRACTS

Type	# Found	Name
Simple	9 250	AutoWallet
	4 635	BasicWallet
	3 634	ConsumerWallet
	46 340	SmartWallet
	1 436	SpendableWallet
	202	TimelockedWallet
MultiSig	11	Argent
	135 457	BitGo
	10 150	Gnosis/ConSensys
	96	Ivt
	3 391	Lundkvist
	995	NiftyWallet
	44 235	Parity/Eth/Wood
	822	TeambrellaWallet
	131	Unchained Capital
Forwarder	1 087 257	BitGo
	2 520	IntermediateWallet
	527	SimpleWallet2
Controlled	2 488 845	Bittrex
Update	2 862	Eidoo
	3 926	LogicProxyWallet
Smart	4 098	Argent
	15 749	Dapper
	1 065	Gnosis

Simple Wallets provide little extra functionality beyond handling Ether and tokens. *MultiSig Wallets* require that m out of n owners sign a transaction before it is executed. Usually the required number of signatures (m) is smaller than the total number of owners (n), meaning that not all owners have to sign. In most cases, the set of owners and the number of required signatures can be updated. *Forwarder Wallets* forward the assets they receive to some main wallet. They may include owner management. *Controlled Wallets* can be compared to traditional bank accounts. They are assigned to customers, who can use them as target of transfers, but the control over the account remains with the bank. Withdrawals are executed by the bank on behalf of the customer. This construction allows to comply with legal regulations that may restrict transactions. Regarding the number of deployments, controlled wallets are the most common type. *Update Wallets* provide a mechanism to update their main features at the discretion of the owner. *Smart Wallets* offer enhanced features like authorization mechanism for arbitrary transactions, recovery mechanisms for lost keys, modular extension of features, or advanced token standards.

V. CREATION AND USE OF WALLETS OVER TIME

Figure 1 depicts the number of wallets created per 100 k blocks (about two weeks) as a stack plot over time. Of the

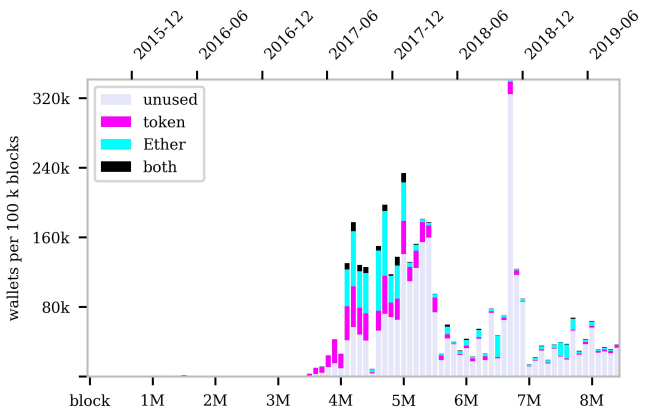

Fig. 1. Creation and usage of all wallets.

3.9 M wallet contracts, 68 % have not been used so far (2.6 M, grey). The other wallets are either used for tokens (529 k, magenta) or for Ether (626 k, cyan), but only a few wallets are used for both (91 k, black). This means that wallet contracts are used either for tokens or for Ether, but rarely for both. Even though most wallets are designed for token management, only 16 % have so far received at least one token. Of the wallets holding tokens, 83 % hold just one type of tokens, 16.5 % hold 2–10 types, and only 0.5 % hold more than 10 different types.

VI. CONCLUSIONS

Our method of computing code skeletons is comparable to the first step for detecting similarities by [2]. Instead of their second step of fuzzy hashing though, we rely on the set of function signatures extracted from the bytecode and manual analysis, as our purpose is to identify wallets reliably. Relying on the interface is in line with the results in [3].

Wallets tend to provide verified source code. The high code homogeneity results from the small number of on- or off-chain factories that generate most of the wallets. Since we manually check the wallet code, our work yields a ground truth of wallets that can be used for evaluating automated tools.

MultiSig wallets were the first to appear in 2015, while controlled and forwarder wallets followed in 2017. Update wallets and smart wallets with a modular design started at the end of 2018. Still, the multiSig wallet seems popular, either as it is or incorporated into smart wallets.

Future work. To detect contract types characterized by their behaviour (like wallets), tools analyzing and classifying automatically the semantics of bytecode are desireable.

REFERENCES

[1] M. Di Angelo and G. Salzer, "Wallet Contracts on Ethereum," *arXiv preprint: 2001.06909*, 2020.

[2] N. He, L. Wu, H. Wang, Y. Guo, and X. Jiang, "Characterizing code clones in the ethereum smart contract ecosystem," *arXiv preprint arXiv:1905.00272*, 2019.

[3] M. Fröwis, A. Fuchs, and R. Böhme, "Detecting token systems on ethereum," in *International conference on financial cryptography and data security*. Springer, 2019.