Towards a First Step to Understand the Cryptocurrency Stealing Attack on Ethereum

Zhen Cheng∗
Zhejiang University
iimp@zju.edu.cn

Xinrui Hou∗
Xidian University
xrhou@stu.xidian.edu.cn

Runhuai Li
Zhejiang University
21821327@zju.edu.cn

Yajin Zhou†
Zhejiang University
yajin_zhou@zju.edu.cn

Xiapu Luo
The Hong Kong Polytechnic University
csxluo@comp.polyu.edu.hk

Jinku Li
Xidian University
jkli@xidian.edu.cn

Kui Ren
Zhejiang University
kuiren@zju.edu.cn

Abstract

We performed the first systematic study of a new attack on Ethereum to steal cryptocurrency. The attack is due to the unprotected JSON-RPC endpoints existed on Ethereum nodes that could be exploited by attackers to transfer Ether and other ERC20 tokens to attackers-controlled accounts. This study sheds light on the attack, including malicious behaviors involved and profits of attackers. Specifically, we first designed and implemented a honeypot that could capture real attacks in the wild. We then deployed the honeypot on the Internet and reported the results of the collected data in a period of six months. In total, our system captured more than 308 million RPC requests, coming from 1,072 distinct IP addresses. We further grouped attackers into 36 groups with 59 distinct Ethereum accounts. Among them, attackers from 34 groups were stealing the Ether, while other 2 groups were targeting ERC20 tokens. The further behavior analysis showed that attackers were following a three-steps pattern to steal the Ether. Moreover, we observed an interesting type of transaction called zero gas transaction, which has been leveraged by attackers to steal ERC20 tokens. At last, we estimated the overall profits of attackers. To engage the whole community, we will release the dataset of all captured attacks by our system.

1 Introduction

The Ethereum network [39] has attracted lots of attentions. Users leverage this platform to transfer the Ether, the official cryptocurrency of the network, or build DApps (decentralized applications) using smart contracts. This, in turn, stimulates the popularity of the Ethereum network.

However, this popularity also attracts another type of users, i.e., attackers. They exploit the insecure setting of the Ethereum clients, e.g., the Go-Ethereum [7] and Parity [14], to steal the cryptocurrency. These clients, if not properly configured, will expose a JSON-RPC endpoint without any authentication mechanism enforced. As a result, it could be remotely reached by attackers to invoke many privileged methods to manipulate the Ethereum account, on behalf of the account holder who is using the client 1. Though we have seen spot reports about the behaviors of stealing Ether by hackers [13,17], there is no systematic study of such an attack. In order words, not enough insights were provided on the attack, e.g., detailed malicious behaviors, attacking strategies, and the potential profits gained by attackers.

To this end, we performed a systematic study to understand the cryptocurrency stealing attack on Ethereum. The purpose of our study is to shed light on this attack, including malicious behaviors involved and profits of attackers. Our study is based on real attacks captured by our system, and transactions occurred.

Specifically, we design and implement a system called Siren. It consists of a honeypot that listens on the default JSON-RPC port, i.e., 8545, of Ethereum clients, and accepts any incoming requests. To make our honeypot a reachable and valuable target for attackers, we register this honeypot as an Ethereum full node on the Internet and prepare a real Ethereum account that has Ethers inside. Moreover, in order to implement an interactive honeypot, we use a real Ethereum client (Go-Ethereum in our work) as the back-end, and redirect all the incoming RPC requests to it and execute them on the client, except for the ones that may cause damage to our honeypot. During this process, our honeypot logs all the information of the request, including the name of the method the attacker intends to invoke, and all the parameters. For instance, our honeypot can log the account addresses that the attackers intend to transfer the Ether into. We call these accounts as malicious accounts, which are controlled by attackers. To estimate profits of attackers, we further crawl all the transactions from the Ethereum network and analyze them to identify suspicious accounts, which have accepted Ethers from malicious accounts. Though we are unaware

1Co-first authors with equal contribution.
†Corresponding author.

Note that, the RPC interface is intended to be used with proper authentication.
about the real owners of these suspicious accounts, they are most likely to be connected with attackers since transactions exist from malicious accounts to them. We then estimate profits of attackers by calculating the income of malicious and suspicious accounts.

Based on the data collected in a six-month period, we have performed a detailed analysis of the captured attacks, and report some interesting findings in the following.

- **Attacks captured** During the six-month period, our system captured 308.66 million RPC requests from 1,072 distinct IP addresses. Among these IP addresses, 9 of them are considered as the main source of attacks, since they count around 83.8% of all requests. One particular IP address 89.144.25.28 sent the most RPC requests, with a record of 101.73 million requests in total. In order to hide their real IP addresses, attackers were leveraging the Tor network [22] to launch attacks. We also observed that some IP addresses from universities worldwide were probing our honeypot, though none of them were invoking methods to steal cryptocurrency. Most of these IP addresses are from the PlanetLab nodes [15].

- **Cryptocurrency targeted** We group attackers based on the IP addresses and the Ethereum accounts that attackers are transferring the cryptocurrency into. By doing so, attackers are grouped into 36 clusters with 59 distinct malicious accounts. Among them, attackers from 34 groups were stealing the Ether, while other 2 groups were targeting ERC20 tokens.

- **Steal Ether** We observed that attackers were following three steps pattern to steal Ether. They first probed potential victims, and then collected necessary information to construct parameters used in the attack. After that, they would launch the attack, either by continuously polling the account state to wait for the account being unlocked, or by brute force to crack the user’s password to unlock the account.

- **Steal ERC20 tokens** Besides the Ether, attackers were also targeting ERC20 tokens. We first observed an interesting type of transaction called zero gas transaction, in which the sender of a transaction does not need to pay any transaction fee. Then we found attackers were leveraging this type of transaction to steal the tokens from *fisher* accounts that intended to scam other users’ Ether, and exploiting the AirDrop mechanism to gain numerous bonus tokens.

- **Profits of attackers** To estimate profits of attackers, we identify suspicious accounts that are connected with malicious accounts. We add the income of malicious and suspicious accounts to estimate profits.

**Contributions** In summary, this paper makes the following main contributions:

- We designed and implemented a system that can capture real attacks of stealing cryptocurrency through unprotected JSON-RPC ports of vulnerable Ethereum nodes.

- We deployed our system and reported attacks observed in a period of six months.

- We reported interesting findings based on the analysis of collected data. We will release the dataset to the community to help propose better methods to detect attackers in Ethereum.

The rest of the paper is structured as the follows: we introduce the background information in Section 2 and present the methodology of our system to capture attacks in Section 3. We then analyze the attack in Section 4 and estimate profits in Section 5, respectively. We then discuss the limitation of our work in Section 6 and related work in Section 7. We conclude our work in Section 8.

## 2 Background

In this section, we will briefly introduce the necessary background about the Ethereum network [39] to facilitate the understanding of this study.

### 2.1 Ethereum Clients and JSON-RPC API

An Ethereum node usually runs a particular client software. There exist several clients, e.g., Go-Ethereum and Parity. Both clients support remote procedure call (RPC) through the standard JSON-RPC API [4]. When these clients start with a special flag, it will listen on a specific port number (e.g., 8545), and accept RPC requests *from any host* through the HTTP protocol *without any authentication*. After that, functions could be remotely invoked on behalf of the account holder of the client, including privileged ones to send transactions (or transfer cryptocurrency). Note that, though the Ethereum network is a P2P network, attackers can discover and reach vulnerable Ethereum nodes directly through the standard JSON-RPC API.

### 2.2 Ethereum Accounts

In Ethereum, there exist two different types of accounts. One is externally owned account (EOA). An EOA account can transfer the Ether, the official currency in Ethereum, to another account. This account can also deploy a smart contract, which in turn creates another type of account, i.e., the smart contract account. A smart contract is a program that executes exactly as it is set up to by its creator (the smart contract developer). In Ethereum, the
smart contract is usually programmed using the Solidity language [16], and executes on a virtual machine called Ethereum Virtual Machine (EVM) [39]. Both accounts are denoted in a hexadecimal format. For instance, the account address 0x6ef57be1168628a2bd6c5788322a41265084408a denotes an (attacker’s) EOA account, while the address 0x87c9ea70f72add5a12bc6155a30e047cf2acad798 denotes a smart contract.

ERC20 tokens [1] are digital tokens designed and used on the Ethereum platform, which could be shared, exchanged for other tokens or even real currency, e.g., US dollars. The community created standards for issuing a new ERC20 token. For instance, the token smart contract should implement a function called transfer() to transfer the token from one account to another, and a balanceOf() function to query the balance of the token. The values of ERC20 tokens vary for different tokens at different times. For instance, the market capital of the Minereum token [9] was more than 7 million US dollars [10] in August, 2017, and is around 40,000 US dollars in March, 2019.

### 2.3 Transactions

Transactions can be used to transfer Ether, or invoke a function of a smart contract. Specifically, the to field of a transaction denotes the destination, i.e., an EOA account or a smart contract. For a transaction to send the Ether, other fields including gas and gasPrice specify the gas limit and the gas price of the transaction. Listing 1 (Section 4 on page 5) shows a real transaction to send ether to 0x63710c26a9be484581dcac1aacdd95ef628923ab, a malicious account captured by our system. If the transaction is used to invoke a function of a smart contract, then the data field specifies the name and parameters of the function to be invoked. Note that, a function is identified by a function signature, i.e., the four bytes hash of the Keccak hash of the canonical expression of the function prototype, including the function name, the parameter types. Listing 2 and 3 (Section 4 on page 6) shows the data field and the invoked function signature and its prototype.

Sending a transaction consumes gas. It is the name of the unit that measures the work needs to be done. It is similar to the use of liter of fuel consumed when driving a car. The actual cost of sending a transaction (transaction fee) is calculated as the product of the consumed gas and the current gas price. The gas price is similar to the cost of each liter of fuel that you are paying for filling up your car. The smallest unit of the Ether is Wei. A Gwei consists of a billion Wei, while an Ether consists of a billion Gwei. The number of the gas consumed for transferring the Ether is 21,000. However, the gas for executing a function of a smart contract is determined by executed instructions of the function.

The transaction fee is paid by the sender, and the miner who is responsible for packing transactions and executing smart contract instructions will receive the fee. To earn a higher transaction fee, miners tend to pack the transaction with a higher gas price. Specifically, the sender of a transaction can specify the gas price in the field gasPrice (Section 4 on page 5). Setting a high gas price could boost the chance of the transaction being packed. We have observed a trend of higher gas price in the transactions to steal Ether (Figure 3 on page 8).

There are multiple RPC methods that could be remotely invoked to send (or sign) a transaction, including eth_sendTransaction and eth_signTransaction. Note that the sender’s account needs to be unlocked before sending a transaction. This involves the process to enter the user’s password. Otherwise, the invocation of these methods will fail. In other words, in order to successfully send a transaction to steal Ether, the sender’s (or victim’s) account should be unlocked. In Section 4.3, we will show that attackers will continuously poll the account until the account is unlocked by users, or launch a brute-force attack using a predefined dictionary of popular passwords.

### 3 Methodology

Figure 1 shows the architecture of our system. In order to capture real attacks towards Ethereum nodes with unprotected HTTP JSON-RPC endpoints, we design and implement a system called Siren. Our system consists of a honeypot that listens on the default JSON-RPC port, i.e., 8545. Any attempt to connect to this port of our honeypot will be recorded. Our system forwards the request to the back-end Ethereum client(1) and returns the results (2). The Ethereum client used in our system is the Go-Ethereum 2. Our honeypot logs the RPC requests with parameters from the attacker, and then imports them into a database(3). To further estimate profits of attackers, we first crawl transaction records from the Ethereum network (3), and further identify suspicious accounts that are connected with attacker’s accounts. Our system combines the transaction records (3) of the mali-

---

2This is the most popular Ethereum client, with a command line tool called Geth. In this paper, if not otherwise specified, the Ethereum client we discussed is the Go-Ethereum.
ocious accounts captured by our honeypot and the suspicious accounts to generate the final results (Θ).

3.1 Ethereum Honeypot

In order to capture real attacks and understand attacking behaviors, we build a honeypot. It can interact with JSON-RPC requests that invoke APIs for malicious intents, e.g., transferring Ether to an attacker controlled malicious account. Our honeypot logs the information of the API invocation, including the method name, the parameters and etc., for later analysis. Moreover, our system confines the attacker’s behaviors, so that APIs invoked by attackers cannot cause real damages to our honeypot.

To this end, we design a front-end/back-end architecture for our honeypot. Specifically, we implement a frontend that listens to the 8545 port and accepts any incoming HTTP JSON-RPC request from this port. We also have a real Ethereum client in the back-end, which runs as a full Ethereum node. This client accepts any local JSON-RPC request from the frontend. That means our real Ethereum client is not publicly available to the attackers. If the invoked API is inside a predefined whitelist, our frontend will forward the request to the back-end client, and then forward the response to the attacker. By doing so, the system protects the Ethereum node from being actually exploited, while at the same time, facilitates the information logging of the requests since all the requests need to go through the front-end.

However, there are still some challenges that need to be addressed to make the system effective. For instance, the honeypot should behave like a real Ethereum node. Otherwise, attackers could be aware the existence of our honeypot and do not perform malicious activities. In the following, we will illustrate the ways that our system leverages to attract attackers, and how the honeypot works.

Respond the probe requests The default port number of the HTTP JSON-RPC service of an Ethereum node is 8545. Before launching a real attack, attackers usually send probe requests to check whether this port is actually open. For instance, the attacker invokes the web3_clientVersion method to check whether it is a valid Ethereum node. The front-end of our honeypot accepts any incoming JSON-RPC request, and responds with valid results, by relaying responses from the back-end Ethereum client.

Advise the existence of our Ethereum node In order to capture real attacks, our system needs to attract attackers. One option is we passively wait for attackers by responding to the probe request. However, this strategy is not efficient, since the chance our honeypot is happened to be scanned is low, given the fact of a really large space of valid IP addresses. To make our Ethereum node (or our honeypot) visible to attackers, we register it on public websites that provide the list of full Ethereum nodes. The original purpose of this list is to speed up the discovery process of Ethereum node in the P2P network, since the list can be directly imported into an Ethereum client. However, this list also provides valuable information to attackers, since they can find potential targets without performing time- and resource-consuming port scanning process. It turns out this strategy is really effective. Our honeypot receives incoming probe requests shortly after we registered it on the list.

Pretend as a valuable target The main purpose of the attack is to steal the cryptocurrency. In order to make the attacker believe our honeypot is a valuable target, we create a real Ethereum account with the address 0xa33023b7c14638f3391d705c938ac506544b25c3 and transfer some amounts of Ether into this account. Since the Ethereum network is a public ledger, the amount of the Ether of our account is a public knowledge, and could be obtained by querying on the network. Our honeypot returns this account address to attackers if they invoke the eth_accounts method to get a list of accounts owned by our Ethereum node. We also return the real amount of Ether inside this account to attackers if the eth_getBalance method is invoked.

Emulate a real transaction After obtaining the information of the account owned by our Ethereum node and the balance of this account, attackers tend to steal the Ether by transferring it to accounts they controlled (malicious accounts). For instance, they could invoke the eth_sendTransaction method, which returns the hash value of a newly-created transaction. Attackers could check the return value of the method invocation to get the status of Ether transfer. To make the attacker believe that the transaction is being processing, while not actually transferring any Ether from our account, we do not actually execute the eth_sendTransaction method on our Ethereum node. Instead, we log the parameters of this method invocation, and return a randomly generated hash value to the attacker.

Log RPC requests Our honeypot logs the attacker’s invoked methods, including the method name, parameters, along with the metadata of the attack, such as the IP address and the time. All the data will be saved into a log file, which will be imported into a MongoDB database [11].

3.2 Data Collection and Analysis

After capturing attacks and malicious account addresses, we will estimate profits gained by attackers. Our system leverages transactions launched from these accounts to find more attacker-controlled accounts. For this purpose, we crawl the transaction history from the Ethereum network.

Our system first downloads all Ethereum transactions, and then uses a Python script to import them into a Greenplum database [8]. By leveraging this database, we can easily query transactions by issuing a SQL statement, a more convenient way for looking up interesting transactions. By combining the data captured from our honeypot and transactions col-
Figure 2: The number of daily RPC requests and distinct IP addresses captured by our honeypot. The seven red circles mean that the data in these days is incomplete, either because our system was accidentally shut down, or because the network is not stable.

lected from the Ethereum network, our system generates final analysis results.

4 Attack Analysis

In this section, we will illustrate the data we collected, the grouping of attackers based on their behaviors, and captured attacks to steal cryptocurrency, including the Ether and ERC20 tokens.

4.1 Data Overview

We deployed our system on a virtual machine in the cloud, and collected the data in six months, i.e., from July 1 to August 31 in the year 2018, and November 1, 2018 to February 28, 2019. Unfortunately, there are three days (from July 24 to July 26) when the virtual machine was accidentally shut down, and we did not have data in these days, and four days (July 14, 15, 27 and 30) when the network is not stable and the data is incomplete. Figure 2 shows the number of daily RPC requests and distinct IP addresses.

In total, our system observed 308,66 million RPC requests from 1,072 distinct IP addresses. In average, we received 1.72 million RPC requests each day (excluding the incomplete data.) In terms of IP addresses, the average daily number is 62. This number reached its peak value (142) on December 24.

Among these RPC requests, 9 different IP addresses are main sources of attacks, given the fact they contribute most of RPC requests in our dataset. These 9 IP addresses sent around 258.70 million requests in total, which counts around 83.8% of all requests. It’s worth noting that, one IP address 89.144.25.28 sent the most RPC requests. It sent 101.73 million requests in total, accounting for 33.0% of the requests we received. We believe such an aggressive behavior is to increase the possibility of stealing the Ether, since the time window to transfer the Ether only exists when the user account is unlocked. We also observed that the attack from this IP address ceased in several days, e.g., from August 15 to 17.

**RPC requests from universities worldwide** Interestingly, we found some RPC requests are from IP addresses that belong to universities. Specifically, our honeypot received requests from 66 IP addresses of 39 universities in 13 countries or regions. Among them, 37 IP addresses are from universities in the USA. For instance, two IP addresses (146.57.249.98 and 146.57.249.99) captured by our honeypot belong to the University of Minnesota. We further used the reverse DNS lookup command to obtain the domain name associated with these IP addresses. It turns out that all of them are associated with the PlanetLab [15]. For instance, the domain names of the previous two IP addresses are planetlab1.dtc.umn.edu and planetlab2.dtc.umn.edu, respectively. Requests from these IP addresses are not performing malicious activities, e.g., transferring Ether to other accounts. Most of them are merely probing our honeypot for information collection, e.g., invoking eth_getBlockByNumber. Though the exact intention of collecting such information is unknown, we believe these requests are mainly for a research purpose.

**Abuse of the Tor network and cloud services** We found attackers are leveraging the Tor network and cloud services to hide their identities. For instance, some IP addresses belong to popular cloud services, e.g., Amazon, DigitalOcean and etc. Among the 1,072 distinct IP addresses, 370 of them are identified as Tor gateways [22] (299 of them performed malicious behaviors, e.g., trying to steal Ether from our honeypot.) All these IP addresses belong to the second group (Section 4.2.) They are from 104 different ISPs in 39 countries. Using the Tor network to hide the real IP addresses make the tracing of attacks more difficult.

4.2 Grouping Attackers Accounts

After obtaining the data, our next step is to group attackers. However, due to the anonymity property of the Ethereum network, it is hard to group them based on their identities. In this paper, we take the following ways to group attackers.

Listing 1: The captured attack and the associated account address in the to field.
we monitor the to way to group attackers. Note that, since attackers are using we will put them into the same group. This is the most direct various ways to hide their own IP addresses, we further changing this address, the attacker can obtain the reward based) account of a miner. This is the address that will be is used to change the etherbase (or coin- eypot is still considered as a suspicious action since it send the transaction, executing this method on our hon- gas consumption of a transaction and does not actually that, though the methods directly transfer Ether to other accounts, and other two eth_sendTransaction and eth_signTransaction. Attackers use this dress of a transaction to transfer Ether. Attackers use this method to transfer (steal) the Ether to their controlled accounts. For instance, the parameter to the group attackers. We found that the value of the attacker-controlled account address. In our system, we monitor the to field in the parameters of the methods eth_sendTransaction and eth_signTransaction that directly transfer Ether to other accounts, and other two methods eth_estimateGas and miner_setEtherBase. Note that, though the eth_estimateGas is used to estimate the gas consumption of a transaction and does not actually send the transaction, executing this method on our honeypot is still considered as a suspicious action since it usually follows a real transaction afterwards. The method miner_setEtherBase is used to change the etherbase (or coin- base) account of a miner. This is the address that will be rewarded when a new block is mined by the miner node. By changing this address, the attacker can obtain the reward Ether, on behalf of the miner.

Third, we indirectly retrieve attackers’ account addresses, and use them for grouping. This is the case when attackers steal ERC20 tokens by calling the standard transfer () method defined in the ERC20 token standard [1]. Their addresses are not in the parameters of the transaction. However, we can obtain them by retrieving parameters of the smart contract method invocation. In the following, we will use a real example to illustrate the steps. Listing 2 shows a captured attack of the RPC request to invoke the method eth_sendRawTransaction. We first decode the params field to obtain the to field (its value is 0x1a95b271b0535d15f4a49932da3b1a6252496). It turns out that this address is not an EOA account, but a smart contract address of the Minereum token [9], whose peak market capital was more than 7 million US dollars in August 2017 [10]. We further decode the data field to retrieve the invoked function in the smart contract and its parameters. The result is shown in Listing 3. We can see the malicious account address that receives the stolen ERC20 token is 0x0fe07dbd07ba4c1075c1db97806ba3c5b113cee0.

Based on previous strategies, we group attackers into 36 groups. The malicious account addresses of each group, the number of IP addresses and the number of RPC requests are shown in Table 1. Among them, attackers from 34 groups (from the group 1 to the group 34) were stealing Ether and other two groups were targeting ERC20 tokens (the group 35 and 36). In the following, we will present the detailed results of attackers’ behaviors to steal Ether and ERC20 tokens.

4.3 Analysis of Ether Stealing

The attackers from the group 1 to the group 34 are stealing Ether. They are following a three-step pattern to perform the attack.

Step 1 - Probing potential victims The first step to launch the attack is to locate potential victims that have insecure HTTP JSON-RPC endpoints. Attackers could obtain potential victims by downloading a list of full Ethereum nodes, or performing a port scanning process to find the machines with target port number (8545 in our study) opening. Then attackers issue RPC requests to determine whether the victim is an Ethereum node or even a miner node that will be useful to send a zero gas transaction (we will discuss this type of transaction in Section 4.4).

The most used commands for probing is shown in Table 2. Specifically, net_version is used to identify the client’s...
Table 1: The result of grouping attackers.

| # | Addresses | # of IP | # of RPC calls |
|---|-----------|---------|----------------|
| 0 | 0xa141e6e12e54c5e13fe651a8fe9b269f4ec3d0 | 57 | 72,915,681 |
| 0xe64c3c7e67565c717c7b5c85bfef437b192a | | |
| 0xe65f7be16282a2b2d0c678722e41265084408a | | |
| 0xe70f7411c1874d2407c6290de0ae0d2ff4d8 | | |
| 0xe611268ccf5c8104ace4817fe5a6e6e6aa88d080e | | |
| 2 | 0x5f8106c85524a2e3a63e92876919de09a1ad54af0f04 | 309 | 363,860 |
| 0xf3ca9ba81b9156d307e6469af5783b70e363b586 | | |
| 0x8acada0622f79b900c00ed0f253d23fb0931 | | |
| 0xab2950c17f470422516aff953a3abc10b9a588a5 | | |
| 0x9de44d0b86e43054246f59e3a2af6cc2847 | | |
| 0x89ab57887b573eb4dd9f15fc03a2d8323c50670 | | |
| 0xe412f7324492ed5eac30300e224053f11326a | | |
| 3 | 0x66f1587a5f729f2cc9ee3c6a117e654bc8ceacedf | 14 | 13,315,318 |
| 0x23d42c42103e239a9a9a007b6076f6878eb3d7 | | |
| 0x7269a0784a031a2e8070f65f6e87837e3e376 | | |
| 4 | 0x046b3e03f2e0f6c5b30e530b76466a14412a0 | 1 | 101,751,595 |
| 0x63710c26429e4a84581dca11ac99d5e619234a | | |
| 5 | 0xb0e5c6f61e4612470392eb98b73d76019743b3c0 | 6 | 326,154 |
| 6 | 0x1a086b35a94f158e792a3ed860f72f00430e | 3 | 6,791,438 |
| 0x73b4c702950f0080f85feeb36856a82502d6e | | |
| 0xa3f4778480e59956d4040607c16f77f73c8ca | | |
| 0xec13873e54d7f93ce33b296bad0c6553a256cb | | |
| 7 | 0x2c194e61b5796e37f2c81296e19a55461f22526da | 1 | 7,750,800 |
| 0x86e25239b9f617b2eb0e5d67c67e82e264e2a6c | 2 | 3,569,924 |
| 0x9ff871039e4f1582f6047903c722ee2e9209ee | 1 | 3,522 |
| 0x263059f99464589588d2604701e39e66e62 | | |
| 11 | 0x2c5239b4c16f6e5f1730c651c1e6b85f2e1c09 | 618 |
| 0x5e93b9e81e60e673076e6a4f79f582b70e430c7b | 1 | 3 |
| 0x5e732b73a919c2eb8ee3c008b04566a4a7581b1 | | |
| 0x5c80e4b581b919999c989c90970e58826f898 | 3 | 80 |
| 0x5e47b4bca1fa08d8f95f3106e2ce0a88e8b4 | 274 |
| 0x977943c5a1e6a9a96e8f84c42f59a9be36060a | 3 | 235,213 |
| 0x9d6df59856efc86f405f384560579e7116d4 | | |
| 0x2e2a4347035b7a0d2792388550331cecb17f688 | 11 | 2,146,017 |
| 0xc3b1eab6c7cc3b06f262652bc8e211f1be7f8ae | | |
| 0xe1286b577284271929b0d65265455941b5b | 1 | 277 |
| 0xb744df573d7210999efecb007062ed7b770a102 | 2 | 237,481 |
| 0x6e0a3096bc1499bd424ca6c56f8d9c95c5bfe60 | 2 | 59,942 |
| 0x2c002e3a09267492970d253c435a7c87d4c474 | | |
| 0x333437fbbf5c1f93239f7940089bf1f4f6dc93f0e2dfe | | |
| 0x88a405a9c103331c561cdaf905e52e1fa0bb | | |
| 0xe6f262ed880c4d0f3783e4de39390609c5a72 | | |

Table 2: Most used commands for probing.

| Command | # of IP addresses | # of RPC requests |
|---------|-----------------|-----------------|
| net_version | 122 | 4,822,620 |
| rpc_modules | 81 | 3,815 |
| web3_clientVersion | 103 | 4,495,312 |
| eth_getBlockByNumber | 325 | 1,190,445 |
| eth_blockNumber | 225 | 27,019,866 |
| eth_getBlockByHash | 214 | 1,633 |

Table 3: Commands used for preparing attacking parameters.

| Command | # of IP addresses | # of RPC requests |
|---------|-----------------|-----------------|
| eth_accounts | 615 | 27,040,164 |
| eth_coinbase | 64 | 87,442 |
| personal_listAccounts | 11 | 95 |
| personal_listWallets | 5 | 173,243 |
| gas_price | 6 | 63,133 |
| eth_getBalance | 493 | 93,385,372 |
| | 6 | 2,411,504 |
Gas Price (Gwei)

• value: This is the value of Ether that will be transferred into the to_address. In order to maximize their income, the attacker tends to transfer all the Ether in the victim account, leaving a small amount to pay the transaction fee. In order to get the balance of the victim’s account, the method eth_getBalance is used by attackers.

• gasPrice: The attacker could set a high gasPrice to increase the chance of the transaction being executed (or packed into a block by miners.) For instance, we found that the attacker (0x21bdc4c2f03e239a59aad7326738d9628378f6af) tends to use a much higher gasPrice in the transaction to steal Ether. Figure 3 shows the gas price of transactions from attackers and normal users. We will illustrate it later in this section.

Step 3 - Stealing Ether In order to successfully send a transaction, it needs to be signed using the victim’s private key. However, the private key is locked by default, and a password is needed to unlock the keyfile. We observed two different behaviors that are leveraged by attackers to solve this problem.

• Continuously polling: Attackers continuously invoke the methods, i.e., eth_sendTransaction or eth_signTransaction in the background. If a legitimate user wants to send a transaction at the same time, then he or she will unlock the account by providing the password. This leaves a time window that the attacker’s attempt to get a signed transaction could succeed.

However, in order to successfully launch the attack, there are still two challenges. First, the time window is really small. Attackers should happen to invoke the methods to send the transaction at the same time when users are unlocking the account. To increase the chance of a successful attack, the operation to send the transaction should be very frequent. That’s the reason of our observation that some attackers are repeatedly invoking the previously mentioned methods at a very high frequency, nearly 50 requests per-second. Second, since the attacker is sending the transaction at the same time with the user (i.e., when the user is unlocking his account), his transaction may fail if the user’s transaction is accepted by the miner at first and the remaining balance of the account will not be sufficient for the attacker’s transaction. In order to ensure that his transaction will be accepted by miner in a timely fashion, the attacker will use a much higher value of the gasPrice than normal transactions to bribe miners.

Figure 3 shows the gas price of transactions from attackers and normal users. Specifically, we first calculate the average gas price of captured transactions of attackers from the group 1 to 8 (the solid line in the Figure). Then we calculate the average gas price of transactions of normal users in six months (the dash line in the Figure). The value is around 20 Gwei. It turns out that the gas price from attacker’s transactions is much higher (from 15 times to 4,500 times) than the value of a normal transaction. Setting a higher gas price can increase the speed that their transactions are packed into a block. This strategy is very effective, and we have observed several cases that the transaction with higher gasPrice succeed, while the ones with lower gasPrice failed [20, 21].

• Brute force cracking: Besides the polling strategy, some attackers are leveraging the brute force attack to guess the password of the keyfile. Specifically, they try to unlock the account using the password in a predefined dictionary. Since the Ethereum client does not limit the number of wrong password attempts during a certain time period, this attack is effective if the victim uses a weak password. For instance, the attacker from the group 11 leveraged this strategy and tried a dictionary with more than 600 weak passwords, e.g., qwerty123456, margarita and 192837465. Another attacker from the group 1 took the same way, but only tried one password (ppppGoogle). The reason why this attacker only tried this specific password is unknown. However, we think it may be the default password for some customized Ethereum clients.

Interestingly, we found after a successful try to unlock the account, the attacker will set a relatively long timeout value by invoking the personal_unlockAccount. By doing so, the account will not be locked again in a long time period and the attacker can perform further attack much easier.
4.4 Analysis of ERC20 Token Stealing

Attackers from two groups (group 35 and 36) are targeting ERC20 tokens. ERC20 is a technical standard used by smart contracts on the Ethereum network to implement exchangeable tokens [1]. The ERC20 token can be viewed as a kind of cryptocurrency that can be sold on some markets, thus becoming valuable targets of attackers.

Before illustrating the detailed attacking behaviors, we will first discuss an interesting type of transaction called zero gas transaction, which we observed in our dataset. It exploits the packing strategy of some miners to send transactions without paying any transaction fee. By using this type of transaction, attackers could perform malicious activities to steal ERC20 tokens from the addresses with leaked private key, or exploit the AirDrop mechanism of ERC20 smart contracts to gain extra bonus tokens with nearly zero cost.

**Zero gas transaction** Sending a transaction in the Ethereum network usually consumes gas (Section 2.3). The actual cost is calculated as the product of the amount of gas consumed and the current gas price. The amount of gas consumed during a transaction depends on the instructions executed in the Ethereum virtual machine, while the gas price is specified by the user who sends the transaction. If not specified, a default gas price will be used.

Interestingly, our honeypot captured many attempts of sending transactions with a zero value in the gasPrice field. This brings our attention for a further investigation. We want to understand whether such transactions could be successful, and the intentions for sending such transactions. After performing experiments on the testnet, we found that transactions with a zero gas price received through the p2p network will not be accepted by the miner, and will be discarded as invalid ones. However, if such a transaction is created and launched on the miner node itself (i.e., the node that successfully mines a new block is sending a zero gas transaction), then the transaction will be packed into the block by the miner itself and accepted by the network.

This explains the captured attempts of such transactions by our honeypot. In particular, the attackers were trying to launch zero gas transactions on every vulnerable Ethereum node, in hope that the node is a miner node that is successfully mining a new block. Though the chance looks really slim, we found several successful cases in reality, e.g., the first several transactions in the block 5899499 [3]. Most of the transactions are transferring ERC20 tokens to the address 0x0fe07dbd07ba4c1075c1db97806ba3c5b113cee0, which is a malicious address of the attacker in group 20 captured by our system.

**Attack I: stealing tokens from fisher accounts** The first type of attack is leveraging the zero gas transactions to steal tokens from fisher accounts. In order to understand this attacker, we first explain what the fisher account is in the following.

The fisher account means that some attackers intentionally leak the private key of their Ethereum accounts on Internet. They also transfer some ERC20 tokens to the accounts as the bait. Since the private key of the account is leaked, other users could use the private key to transfer out the ERC20 tokens. However, there is one problem in this process. In order to transfer the ERC20 tokens, the account should have some Ethers to pay the transaction fee. As a result, one may transfer some Ethers into this account, in hope to get the ERC20 tokens. Unfortunately, after transferring the Ether into this account, the Ether will be transferred out to some other accounts immediately by attackers. That’s the reason why such an account is called the fisher account. The main purpose of leaking the private key is to seduce others transferring Ether into the fisher account.

For instance, we found a fisher account whose address is 0xa8015df1f65e1f3d4g1dce1d3s5013031ad25034 [2]. The attacker bought 75,000 ICX (a ERC20 token) as the fishing bait that values around 66,000 US dollars. "Occasionally", the fisher released the private key of that account on the Internet. Anyone who transfers the Ether into this account and hopes to obtain the ICX token will be trapped to lose the transferred Ether.

Interestingly, by leveraging the zero gas transaction previously discussed, attackers could steal the ERC20 tokens in the fisher account. Specifically, attackers could send the transactions to transfer the ERC20 tokens in the fisher account with zero gas price. If the transaction is successful, then the attackers will obtain the ERC20 tokens without any cost.

In our dataset, the user in group 35 (the address is 0x0fe07dbd07ba4c1075c1db97806ba3c5b113cee0) was performing this type of attack. In total, the attacker sent 61,158 RPC requests, stealing 161 different types of ERC20 tokens. The information of the top ten ERC20 tokens this attacker is targeting, including the smart contract addresses, the number of RPC requests and the name of the token, are shown in Table 4. We observed several different IP addresses (62.75.138.194, 77.180.167.78, 77.180.200.1, 92.231.160.88, 92.231.169.137, 95.216.158.152 and etc.) from this attacker.

**Attack II: Exploiting the airdrop mechanism** Airdrop is a marketing strategy that the token holders would receive bonus tokens based on some criteria, e.g., the amount of total tokens they hold. The conditions to send out bonus tokens depend on the individual token maintainer.

We found that some attackers are leveraging the zero gas transaction to obtain the free LEN tokens. Specifically, the LEN token has an airdrop strategy that if a new user A sends any amount of LEN token to the user B, then both A and B will be rewarded with 18,895 LEN tokens. Hence, the attacker could create a large number of new accounts, and then transfer LEN tokens to the attacker’s address (address...
Table 4: The top 10 ERC20 tokens that attackers are targeting.

| ERC20 token addresses | # of RPC requests | Token name |
|-----------------------|-------------------|------------|
| 0x1a95b271b0535d15fa49932dab31ba612b5294e1 | 11,788 | MNE |
| 0xee2e1231b34f73890e92991d55f6090ce17930b92 | 8,998 | DLYC |
| 0xe0775b81a273b355e6a5b7e6240b708701f02027 | 8,099 | BUL |
| 0xbdbd382351fe1b46687fa19d1c650f9941eef89 | 7,735 | SVD |
| 0xc6675daa97475a2580b05a3e08635cf03c8e964bfaf7e4 | 7,359 | TKNL |
| 0x58707ea707f72d555a1b2f1f155a30e047cf2accdf790 | 7,058 | LEN |
| 0x44c95672e663352224032260a4b505016d4a26d | 5,510 | VGS |
| 0x2335202036e99a12c092b5e2e1a9e692658ad6d9 | 4,011 | FDZ |
| 0xc5b813ebc6cfa3c67eb7b79fe90b73eb33f4808d7c | 2,219 | SAT |
| 0x89700d6cd777df52c29ca776a13c3313206eca5 | 1,708 | PMD |

0xffecffe94c3e87987454f2392676ccdb989b268f8 in group 36. By doing so, the new account will receive a bonus token, which will be transferred to the attacker’s account, while at the same time the attacker’s account will also receive the bonus. We observed many attempts of such transactions using zero gas price, with 7,058 different source account addresses and one destination address (the attacker’s address). This transaction does not consume any gas, and the attacker could be rewarded with ERC20 tokens. By using this method, the attacker even becomes a large holder of this token (2.4%) [18].

5 Transaction Analysis

After capturing malicious accounts from our honeypot and analyzing the detailed attackers’ behaviors, we further estimate profits of attackers. Though we can directly get the estimation by calculating the income of malicious accounts, attackers may use other account addresses that have not been captured by our honeypot. We call these addresses that are potentially controlled by attackers as suspicious accounts.

In our system, we take the following steps to detect suspicious accounts. The basic idea is if the attacker transfers the Ether from a malicious account to another account, it is highly possible that the destination account is connected with the attacker. The attacker has no reason to transfer the Ether to an account that has no relationship with him. Note that, the attacker could transfer the Ether to a cryptocurrency market, where he can exchange it with other types of cryptocurrency or US dollars. Hence, we remove the addresses of cryptocurrency market from suspicious accounts in our study using the information queried from the Etherscan website [6].

To this end, we used a similar idea of the taint analysis [36] to find suspicious accounts. Specifically, we treat malicious accounts captured by our system as the taint sources, and propagate the taint through the transaction flows until reaching the taint sinks, i.e., the cryptocurrency markets. We also estimate the attackers’ income when they transfer Ether to cryptocurrency markets. We also remove the addresses with zero profit from the table (e.g., addresses in the group 10), and we do not count the attackers from the group 36 since they are targeting the ERC20 tokens, whose value are hard to estimate due to the dramatic change of the token price. It’s worth noting that, the actual income of attackers is far greater than the estimation by calculating the income of malicious accounts, with the attacker even becoming a large holder of this token (2.4%) [18].

Figure 4: One example of detecting suspicious accounts through the transaction analysis. The house ones are the cryptocurrency markets, the circle one is the malicious account. The box ones without background color are the detected suspicious accounts, while the ones with gray background are unknown accounts.
more than the value shown in the table.

6 Discussion

Though we have adopted several ways to make our honeypot an interactive one, cautious attackers can still detect the existence of our honeypot and do not perform malicious activities thereafter. For instance, the attacker can first send a small amount of Ether to a newly generated address and then observe the return value (the transaction hash) of this transaction. Since the transaction to send Ether in our honeypot does not actually happen, the return value is an invalid one (a randomly generated value). Or the attackers can simply send some uncommon commands and observe the return value to detect the honeypot. Nevertheless, honeypots are always in an arms race with attackers. It is an open research question to propose more effective countermeasures to improve the honeypot.

In this paper, we take a conservative way to detect suspicious accounts and estimate profits of attackers. Specifically, we leverage the knowledge of whether an address belongs to a cryptocurrency market and mark the tainted accounts whose Ether eventually flows into cryptocurrency markets as suspicious. However, the knowledge of the mapping between addresses and cryptocurrency markets may not be complete, since these addresses are manually labelled. Some suspicious accounts may be identified as unknown ones in our work, hence introducing false negatives to our work. Moreover, our estimation is based on the attacker’s addresses collected by our honeypot (in six months). There do exist attackers who are missed by our system, and profits of these attackers are not included in our estimation. We believe the total income of attackers in the wild is much bigger than our estimation.

In this paper, attackers are exploiting the unprotected JSON-RPC interface to launch attacks. Though it is simple to fix the problem by changing the configuration of the Ethereum client, we are surprised by the fact that 7% of Ethereum nodes are still vulnerable. Specifically, we performed a port scanning to the 15,560 Ethereum public nodes [5] and found that around 1,000 of them are reachable through the RPC port without any authentication 3. This fact demonstrated the severity of this problem, and advocates the need to have a better understanding of this issue in the community (the purpose of our work.)

7 Related work

Honeypot Honeypot systems have been widely used to capture and understand the actions and motivations of attackers by capturing malicious activities [24, 27, 34, 37]. For instance, HoneyD [34] is one of the best-known honeypot projects. It can simulate the network stack of many operating systems and arbitrary routine topologies, thus making it a highly interactive one. Collapsar can manage a large number of interactive honeypots, e.g., Honeypot farms. The concept of honeypot has been adopted to detect attacks to IoT devices [25, 26, 30, 33] and mobile devices [40]. Our system is working towards Ethereum clients, which have different targets with previous systems. The general idea of attracting attacks and logging behaviors are similar, but with different challenges.

Security issues of smart contracts One reason that Ethereum is becoming popular is its support for smart contracts. Developers can use contracts to develop decentralized apps (or Dapps), including the lottery game, or digital tokens. Since its introduction, security issues of smart contracts have been widely studied by previous researchers. Atzei et al. systematically analyzed the security vulnerabilities of Ethereum smart contracts, and proposed several common pitfalls when programming the smart contracts [23]. For instance, the stack size of the Ethereum virtual machine is limited, attackers could leverage this to hijack the control flow of the smart contracts. A system called tEthere [29] is proposed to automatically generate the exploits to attack the vulnerable smart contracts. The evaluation showed that among 38,757 unique smart contracts, 815 of them could be automatically exploited.

To mitigate the threats, some tools to analyze the smart contracts [12, 28, 32, 38, 41] or fix the vulnerable contracts [35] have been proposed. For instance, Oyente [31] is a system that can automatically detect smart contracts vulnerabilities using the symbolic execution engine. Moreover, it can make the smart contracts less vulnerable by proposing some new semantics to the Ethereum virtual machine. Sereum [35] automatically fix the re-entrancy vulnerabilities in smart contracts by modifying the Ethereum virtual machine. Erays [41] is a tool to analyze the smart contracts without the requirement of the source code. In particular, it takes the bytecode and then translates it to the high-level pseudocode that is readable for manual analysis. Securify [38] automatically proved the smart contract behaviors as safe or unsafe. Other similar tools to analyze smart contracts include Mythril [12] and Maian [32].

8 Conclusion

In this paper, we performed a systematic study to understand the cryptocurrency stealing on Ethereum. To this end, we first designed and implemented a system that captured real attacks, and further analyzed the attackers’ behaviors and estimated their profits. We report our findings in the paper and will release the dataset of attacks to engage the whole research community.
Table 5: Our estimation of profits of attackers in ether and US dollars. The price of one ether is around 139 US dollars (March, 2019). We remove the addresses with zero profit from the table.

| # | Addresses | Malicious | Plus Suspicious |
|---|-----------|-----------|-----------------|
|   |           | Ether     | USD             | Ether           | USD             |
| 1 | 0x6a141e661e24c5e13fe651da8fe9b269fe43df0 | 116.91 | $16,280.23 | 814.45 | $113,412.00 |
|   | 0x6e4cc37e765bd7c11cc7b5cbfcb5cbfe473b192e | 56.16 | $7,820.34 | 794.67 | $110,657.59 |
|   | 0xef157be118682a2b6dc5788322a41265084408a | 37.79 | $5,261.74 | 1,420.06 | $197,743.19 |
|   | 0x7097f1f1c1847d5207c629d0e0af9df2d9d58 | 281.44 | $39,191.07 | 1,331.74 | $185,444.98 |
|   | 0xe11126c5cf6104ac8f7d401f6e6ea588d86ebbb | 152.26 | $21,201.86 | 1,332.53 | $185,554.52 |
|   | 0x865232bd96f1f2b0ade5f6c67871e26b6e4a86a | 37.75 | $5,256.18 | 1,066.31 | $148,483.43 |
|   | 0xf1871093e4f15bf40d7903cc72ee42e9026ee | 0.00 | $0.69 | 9.34 | $1,300.01 |
| 2 | 0x5fa38ab6991965dd3507e9ad5f958b2e53b3eb5 | 48.24 | $6,716.88 | 94.28 | $13,129.14 |
|   | 0x8cacaf06027b9b00ceada0f6b34b2f39031 | 0.00 | $0.14 | 10.66 | $1,483.80 |
|   | 0x259c7e14f7042251a6a8f9535aaca2ca5a8585ae | 2.53 | $351.64 | 4.18 | $581.92 |
|   | 0xd9ee4d08a86b430544254ff95e32aa6fccc1d3163 | 54.12 | $7,535.80 | 55.72 | $7,759.26 |
|   | 0x88b7d887b7873f7b9ff930a2d62336c0670 | 0.24 | $33.41 | 0.24 | $33.41 |
|   | 0x21946e1eb9767c91c296119e55461f22b2eada | 1.53 | $213.74 | 1.53 | $213.74 |
|   | 0x9781d03182264968d430a4f0579972575d9844d | 50.32 | $7,006.89 | 61.47 | $8,560.18 |
| 4 | 0x04dcb3ed03f82c685c5b2b0b256f7e6a1d241 | 2.38 | $331.13 | 2.38 | $331.13 |
|   | 0x63710c26a9be484581dcac1aacdd95ef628923ab | 19.44 | $2,706.79 | 38.88 | $5,413.47 |
|   | 0x9a08b635a596a1282eadb158792a3ed4b072f00fe | 0.87 | $120.84 | 1.64 | $227.89 |
| 6 | 0x73b4c0725c900f0208bf5eb3685babc502e26 | 1.10 | $153.12 | 1.10 | $153.12 |
|   | 0x21bdc4c2f024124703b9e29b37e650f5bf9d43b2c2 | 0.87 | $120.84 | 1.64 | $227.89 |
| 11 | 0x2c5129bdcc6865e17360c551ec46815ef2e1ec8 | 113.93 | $15,864.25 | 506.86 | $70,580.16 |
|   | 0xe511268ccf5c8104ac8f7d01a6e6ea588d86ebbb | 0.05 | $6.42 | 0.05 | $6.42 |
| 17 | 0x2534703b57b8da27923885503313ecbc17688b | 4.30 | $598.46 | 4.30 | $598.46 |
|   | 0x21bdc4c2f024124703b9e29b37e650f5bf9d43b2c2 | 0.21 | $29.19 | 0.21 | $29.19 |
| 26 | 0x6c65fa17625f922e9c9c6a6117e54c9f45e4f5d | 2,030.19 | $282,704.44 | 2,030.19 | $282,704.44 |
|   | 0x72bf92c0ce2a939a9ad3273682837816af | 357.78 | $49,802.26 | 58,692.91 | $85,178,982.00 |
|   | 0x72bf92c0ce2a939a9ad3273682837816af | 558.32 | $77,746.63 | 59,298.45 | $85,257,309.40 |
| 30 | 0xc6c42a825555f5fe7d421b3cb6b0f7074325c348 | 1.50 | $208.36 | 1.50 | $208.36 |
|   | 0x2e57e7e66db3091a2a663e36ac90615fc81eb | 0.04 | $6.27 | 0.05 | $7.34 |

Total: 4,193.58 | $583,956.23 | 133,273.46 | $18,558,328.61