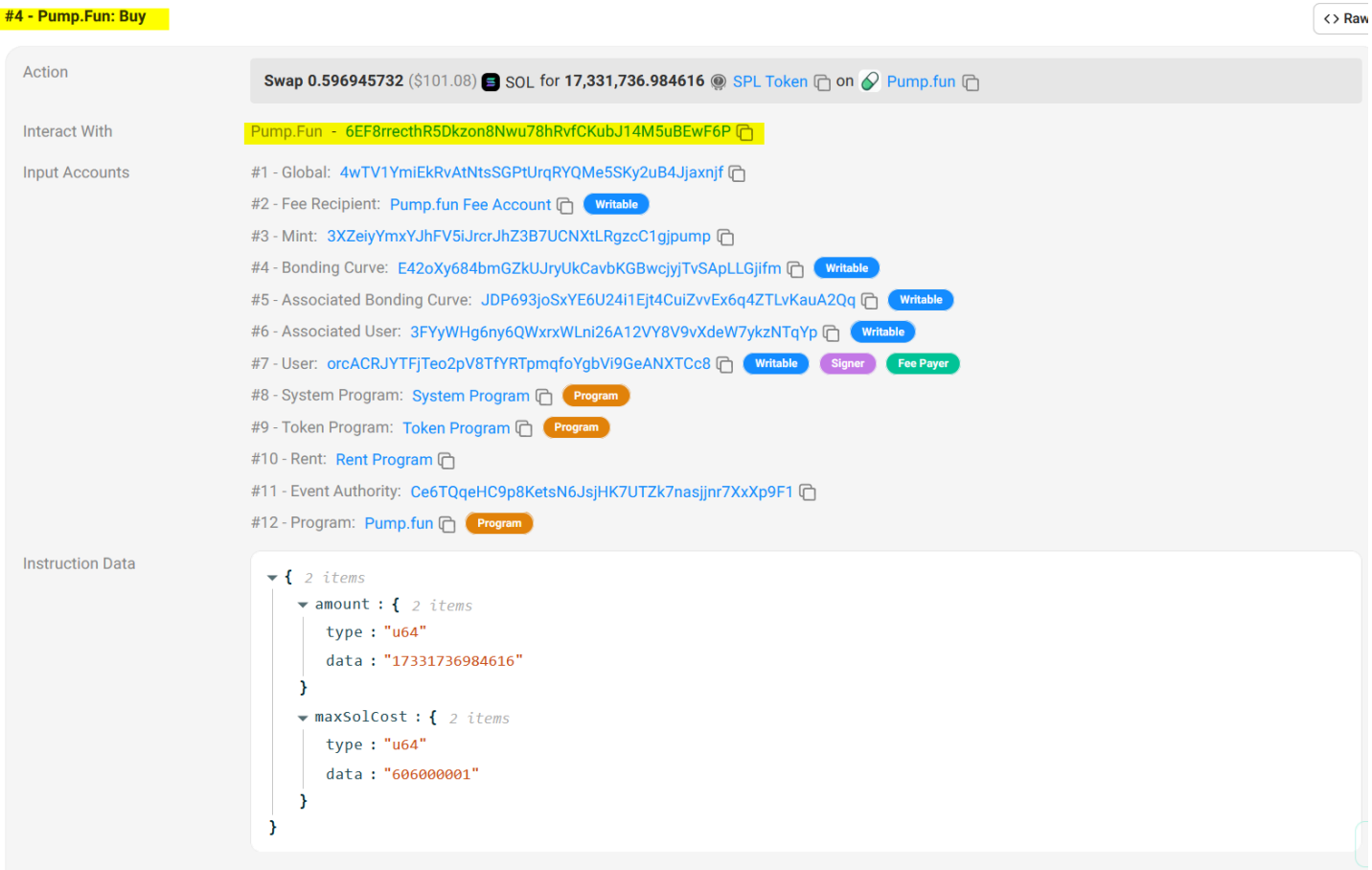
"If You Ain't First, You're Last" Reverse-Engineering a Leading Pump.fun Sniper Bot – Part 2

Nested
Jul 29, 2024 — 4 min read

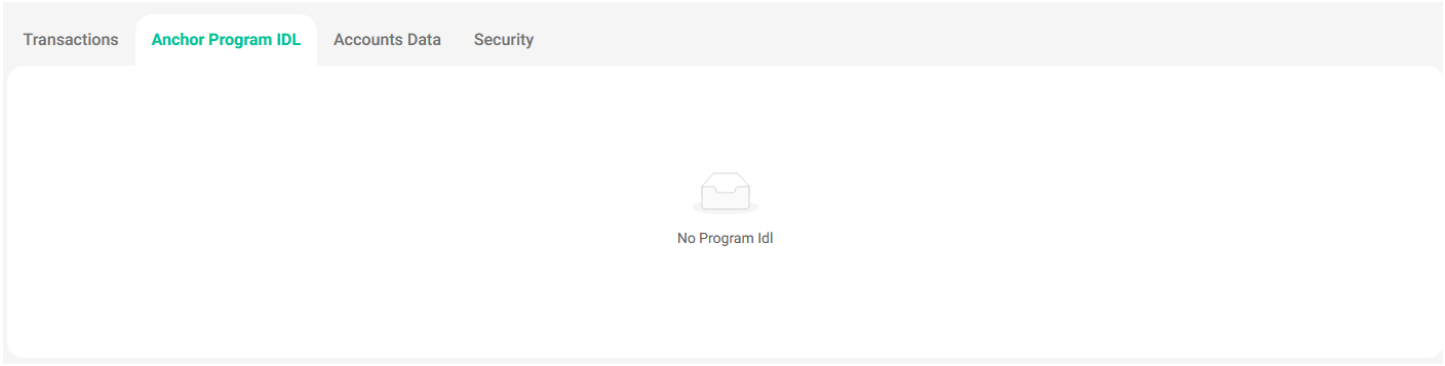
This blog post includes the codebase that @licketyspliket and I used to develop our bot. It is a continuation of the first post in this series, implementing the strategy we previously reverse-engineered and backtested. In this article, I will provide a high-level overview of the various steps and challenges we faced, demonstrate how the bot performed, and discuss potential future improvements.

How do you programmatically interact with Pump.fun?

The first big hurdle we faced was determining how to buy and sell coins on the platform using a language like JavaScript or Python. Looking at a buy transaction on Solscanner, we could see that the transaction uses the Pump.fun contract to perform a buy operation.

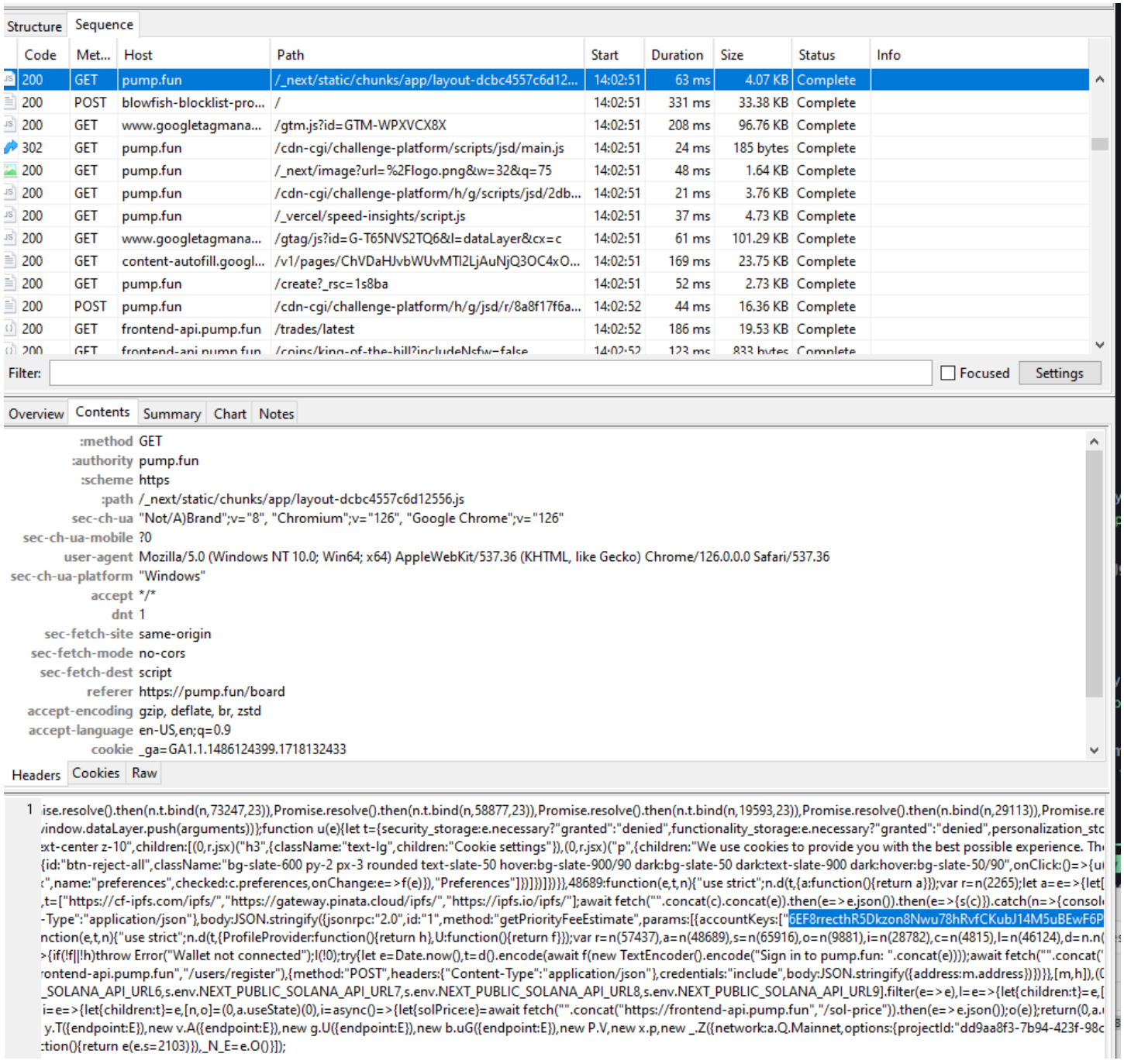


After doing more research into how Solana programs work, we realized that an "Anchor Program IDL" would make it much easier to hook into the program and use its functions. However, no IDL was available on Solscan, so we had to find another way to fetch it.



To accomplish this, I connected Charles Proxy to my Chrome browser and opened Pump.fun. This provided a log of all assets and scripts fetched when the page loaded. Since the browser needs an easy way to interface with the Pump.fun program to mint, buy, and sell coins, it seemed likely that we would find the IDL there.

After searching the logs for the program address, 6EF8..., I found a JavaScript file that looked promising. After beautifying the script and extracting its contents, we discovered it contained the exact IDL we needed.

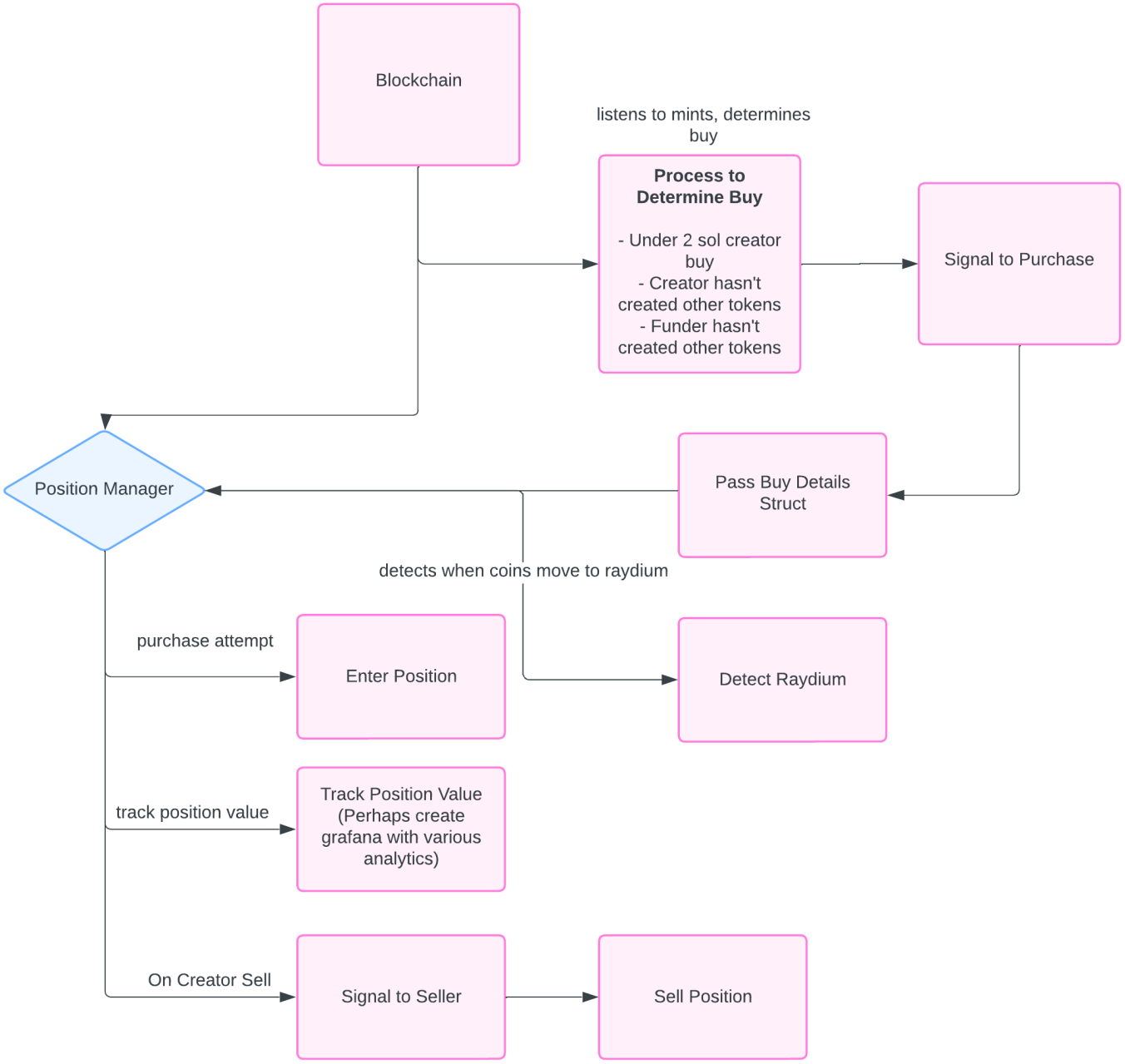

Structure
Sequence
Code Met... Host Path Start Duration Size Status Info
200 GET pump.fun /_next/static/chunks/app/layout-dcbc4557c6d12... 14:02:51 63 ms 4.07 KB Complete
200 POST blowfish-blocklist-pro... / 14:02:51 331 ms 33.38 KB Complete
200 GET www.googletagmana... /gtm.js?id=GTM-WPXVCX8X 14:02:51 208 ms 96.76 KB Complete
302 GET pump.fun /cdn-cgi/challenge-platform/scripts/jsd/main.js 14:02:51 24 ms 185 bytes Complete
200 GET pump.fun /_next/image?url=%2Flogo.png&w=32&q=75 14:02:51 48 ms 1.64 KB Complete
200 GET pump.fun /cdn-cgi/challenge-platform/h/g/scripts/jsd/2db... 14:02:51 21 ms 3.76 KB Complete
200 GET pump.fun /_vercel/speed-insights/script.js 14:02:51 37 ms 4.73 KB Complete
200 GET www.googletagmana... /gtag/js?id=G-T65NVS2TQ6&l=dataLayer&cx=c 14:02:51 61 ms 101.29 KB Complete
200 GET content-autofill.googl... /v1/pages/ChVDaHJvbWUvMTI2LjAuNjQ3OC4xO... 14:02:51 169 ms 23.75 KB Complete
200 GET pump.fun /create?_rsc=1s8ba 14:02:51 52 ms 2.73 KB Complete
200 POST pump.fun /cdn-cgi/challenge-platform/h/g/jsd/r/8a8f17f6a... 14:02:52 44 ms 16.36 KB Complete
200 GET frontend-api.pump.fun /trades/latest 14:02:52 186 ms 19.53 KB Complete
Filter:
Focused
Settings
Overview
Contents
Summary
Chart
Notes
:method GET
:authority pump.fun
:scheme https
:path /_next/static/chunks/app/layout-dcbc4557c6d12556.js
sec-ch-ua "Not/A)Brand";v="8", "Chromium";v="126", "Google Chrome";v="126"
sec-ch-ua-mobile ?0
user-agent Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.0.0 Safari/537.36
sec-ch-ua-platform "Windows"
accept */*
dnt 1
sec-fetch-site same-origin
sec-fetch-mode no-cors
sec-fetch-dest script
referer https://pump.fun/board
accept-encoding gzip, deflate, br, zstd
accept-language en-US,en;q=0.9
cookie _ga=GA1.1.1486124399.1718132433
Headers
Cookies
Raw
1 ise.resolve().then(n.t.bind(n,73247,23)),Promise.resolve().then(n.t.bind(n,58877,23)),Promise.resolve().then(n.t.bind(n,19593,23)),Promise.resolve().then(n.bind(n,29113)),Promise.re
/indow.dataLayer.push(arguments)};function u(e){let t={security_storage:e.necessary?"granted":"denied",functionality_storage:e.necessary?"granted":"denied",personalization_stc
ext-center z-10",children:[(0,r.jsx)("h3",{className:"text-lg",children:"Cookie settings"}),(0,r.jsx)("p",{children:"We use cookies to provide you with the best possible experience. Th
{id:"btn-reject-all",className:"bg-slate-600 py-2 px-3 rounded text-slate-50 hover:bg-slate-900/90 dark:bg-slate-50 dark:text-slate-900 dark:hover:bg-slate-50/90",onClick:()=>{u
:",name:"preferences",checked:c.preferences,onChange:e=>f(e)}),"Preferences"]})]})]})}},48689:function(e,t,n){"use strict";n.d(t,{a:function(){return a}});var r=n(2265);let a=e=>{let[
,t=["https://cf-ipfs.com/ipfs/","https://gateway.pinata.cloud/ipfs/","https://ipfs.io/ipfs/"];await fetch("".concat(c).concat(e)).then(e=>e.json()).then(e=>{s(c)}).catch(n=>{consol
-Type":"application/json"},body:JSON.stringify({jsonrpc:"2.0",id:"1",method:"getPriorityFeeEstimate",params:[{accountKeys:["6EF8rrecthR5Dkzon8Nwu78hRvfCKubJ14M5uBEwF6P
nction(e,t,n){"use strict";n.d(t,{ProfileProvider:function(){return h},U:function(){return f}});var r=n(57437),a=n(48689),s=n(65916),o=n(9881),i=n(28782),c=n(4815),l=n(46124),d=n.n(
>{if(!f||!h)throw Error("Wallet not connected");l(!0);try{let e=Date.now(),t=d().encode(await f(new TextEncoder().encode("Sign in to pump.fun: ".concat(e))));await fetch("".concat("
rontend-api.pump.fun","/users/register"),{method:"POST",headers:{"Content-Type":"application/json"},credentials:"include",body:JSON.stringify({address:m.address})})}},[m,h]),(0
_SOLANA_API_URL6,s.env.NEXT_PUBLIC_SOLANA_API_URL7,s.env.NEXT_PUBLIC_SOLANA_API_URL8,s.env.NEXT_PUBLIC_SOLANA_API_URL9].filter(e=>e),l=e=>{let{children:t}=e,[
i=e=>{let{children:t}=e,[n,o]=(0,a.useState)(0),i=async()=>{let{solPrice:e}=await fetch("".concat("https://frontend-api.pump.fun","/sol-price")).then(e=>e.json());o(e)};return(0,a.
y.T({endpoint:E}),new v.A({endpoint:E}),new g.U({endpoint:E}),new b.uG({endpoint:E}),new P.V,new x.p,new _.Z({network:a.Q.Mainnet,options:{projectId:"dd9aa8f3-7b94-423f-98c
:tion(){return e(e.s=2103)}),_N_E=e.O()}]);

With the IDL in hand, we were able to start placing buy and sell orders.

Developing the bot

Since Lick and I both like working in Go, we chose to build the bot using it. Fortunately, Gagliardetto on GitHub has created some excellent open-source packages for working with the Solana blockchain in Go. Specifically, solana-go provides the main functionality for interacting with the blockchain, and anchor-go allows the bot to use the extracted IDL to interact with the Pump.fun program.

With the strategy from Part 1 of this series in mind, I developed a diagram to outline how I envisioned the bot's operation:


Blockchain
listens to mints, determines buy
Process to Determine Buy
- Under 2 sol creator buy
- Creator hasn't created other tokens
- Funder hasn't created other tokens
Signal to Purchase
Position Manager
Pass Buy Details Struct
detects when coins move to raydium
purchase attempt
Enter Position
Detect Raydium
track position value
Track Position Value (Perhaps create grafana with various analytics)
On Creator Sell
Signal to Seller
Sell Position

Although we never implemented all the features, such as analytics and Raydium monitoring, the diagram served as a helpful guide for handling new mints, from initial pickup to selling the coin.

High-level overview of the bot

Detailing every step of the bot's development would be time-consuming and likely not very interesting, so I encourage interested readers to explore the code provided here for a deeper understanding. However, here's a brief overview beyond the diagram:

The bot has two crucial components: buying and selling handling. Using our dedicated RPC and WebSocket clients, we monitor newly minted coins, gather information about the creator, track the number of coins they purchased, and identify the funder, all within a few hundred milliseconds.

When a buy attempt is triggered, the bot stores the coin and its relevant information in the instantiated `Bot` struct. Simultaneously, it begins monitoring transactions under the creator's Associated Token Account for this coin. If the creator transfers or sells coins, the bot's selling component immediately exits the position. However, as our buy transaction might still be pending or could fail, the bot waits for the buy routine to complete before attempting to sell.

Pitfalls and setbacks

Throughout the bot's development, we encountered a few issues and difficult problems. I've shared a brief overview of some of them below.

- **Reverse-engineering the bonding curve:** This process took a few hours of development to determine the values fetched on-chain from the coin's bonding curve address, allowing us to request the correct amount of coins in our Buy instruction. ChatGPT assisted in rewriting the JavaScript code into Python, which was then converted into Go.
- **Improving transaction confirmation rate and speed of confirmation:** Many developers face issues with Solana transaction landing rates and speeds. To mitigate this, we set up our own dedicated RPC, used free/public RPCs for broadcasting transactions, and used Lick's seamless Jito bundle integration, based on weeaa's jito-go package, significantly improving speed and reliability. The transaction settings we used are available in the repository associated with this blog post.
- **Stale / out-of-sync data from paid RPC services:** Initially, we purchased an RPC subscription from a leading vendor (non-dedicated) and encountered issues with the WebSocket connection being out of sync with the RPC. Additionally, data occasionally arrived seconds after hitting the chain, causing us to miss out on many coins. Operating without a dedicated RPC severely limited the bot's performance.

Did it work?

Overall, the bot could have performed better in production. While its core functionality was solid, even with the addition of Jito bundles and other features, we consistently struggled to be the first entrant into coins. Compared to orcACR's trading activity, we managed to enter roughly 5% of the coins they did, despite our algorithm triggering for many of them.

However, we did achieve first entry into a few coins that reached Raydium, yielding a profit of around 50-60x on those. Despite these successes, we lost on the overwhelming majority of coins we entered and could not consistently turn a profit.

Closing thoughts

Although we didn't achieve the profit margins that traders like orcACR managed with this strategy, working on this project was incredibly fulfilling. Lick and I learned a great deal about building on Solana.

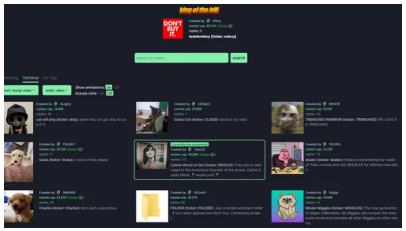
I invested significant time trying to improve the code and enhance our entry rate and transaction confirmation speed, but I couldn't make any meaningful breakthroughs. I think the project would have been much more successful if we could have improved our transaction speed and success rate.

This series, along with the provided code, shares our journey and approach to this project. I also hope to provide a starting point for anyone interested in working on Pump.fun or similar projects. Many GitHub repositories related to Pump.fun contain wallet-drainers or other malicious code, so I wanted to offer something legitimate.

Thank you for reading!

Repository: https://www.github.com/1fge/pump-fun-sniper-bot/

READ MORE



"If You Ain't First, You're Last" Reverse-Engineering a Leading Pump.fun Sniper Bot – Part 1

In early May, I came across the site Pump.fun, where...

By Nested — Jul 20, 2024

Nested Tech Write-Ups

Powered by Ghost

Nested Tech Write-Ups

Articles on automation and web security.

jamie@example.com

Subscribe