12/10/2021

The 0 to 1 guide for MEV

This is a quick and dirty guide to MEV meant to get you up to speed on the latest in the space, nothing more, nothing less. The guide is sorted so that each subsequent topic builds upon previous knowledge, and increases in complexity as topics advance. Obviously skip things you already know. The guide is also nested depending on the level of depth you wish to explore for each particular topic. Each nested child topic represents a further niche concept in the parent topic. Most general Crypto/DeFi topics will just point you to existing high quality material (no need to reinvent the wheel) but more advanced MEV topics (and alpha) will consist of everything I know and what I've learned along the way. Just be warned this is effectively throwing you in the deep end. By the end of this you should be able to build your own MEV Bots. Good luck.

What is Blockchain

Easy (~5min):
https://www.coinbase.com/learn/crypto-basics/what-is-a-blockchain
Medium (~30min)
https://www.investopedia.com/terms/b/blockchain.asp

What is Ethereum

Easy (~5min):
https://ethereum.org/en/what-is-ethereum/
Medium (~1hr):
https://ethereum.org/en/whitepaper/
Hard (~1 day):
https://ethereum.github.io/yellowpaper/paper.pdf

Developer Intro:
https://ethereum.org/en/developers/docs/

Rabbit Hole:
https://ethereum.org/en/learn/

Smart Contracts

Easy:
https://ethereum.org/en/smart-contracts/
Hard:
https://github.com/ethereumbook/ethereumbook/blob/develop/07smart-contracts-solidity.asciidoc#what-is-a-smart-contract

Learn through a Course (~7 Days):
https://cryptozombies.io/

Learn Everything Fast (~1 Day):
https://docs.soliditylang.org/

Learn By Example:
https://github.com/James-Sangalli/learn-solidity-with-examples

Boilerplate Code:
https://docs.openzeppelin.com/contracts/4.x/

What is DeFi

Quick High Level:
https://ethereum.org/en/defi/

If You Want to Read Everything:
https://docs.ethhub.io/built-on-ethereum/open-finance/what-is-open-finance/

ERC-20 Tokens / AKA Most Shitcoins

https://cointelegraph.com/explained/erc-20-tokens-explained

Anyone can deploy an ERC 20 token on the Ethereum blockchain. Depending on gas prices it can cost anywhere from $5-$500 to deploy an ERC-20 token contract. Compared to creating a whole new blockchain from scratch, ERC-20s are the easiest way for anyone to create a cryptocurrency thus making it the most commonly used mechanism to launch shitcoins.

Developer Docs:
https://ethereum.org/en/developers/docs/standards/tokens/erc-20/#top
https://docs.openzeppelin.com/contracts/4.x/erc20
https://eips.ethereum.org/EIPS/eip-20

Find Some Tokens:
https://etherscan.io/tokens

Stablecoins

A stablecoin is a cryptocurrency with the sole purpose of achieving parity with the US Dollar. Stablecoins help increase liquidity in the DeFi space by negating the need to constantly on and off ramp crypto with fiat and allowing an easy mechanism to exchange other crypto assets with dollar based tokens.

There are two broad overarching types of stablecoins. Centralized and Algorithmic stablecoins:

***Centralized stablecoins** consist of a corporation that accepts dollar deposits and mint the corresponding amount of stablecoins for the depositor. Centralized stablecoins have the* *widest adoption by far due to their ease of use, integration with much of the traditional financial system and centralized exchanges. The two most widely used stablecoins are Tether's USDT and Circle's USDC with a combined circulating supply of ~117 Billion USD. However, these centralized stablecoins go against the entire ethos of the trustless decentralized nature of blockchain. There is significant uncertainty regarding the true backing of these stablecoins. Tether is notoriously opaque and has never produced an audited report regarding the backing of it's stablecoins. Read more here. Circle, while nowhere near as suspect still has questions regarding the assets backing its coin and should likewise be treated with caution.*

***Decentralized Stablecoins** represent the purest form of a dollar pegged cryptocurrency in DeFi. At its core decentralized stablecoins attempt to achieve dollar parity by having users deposit collateral and mint the protocol's native stablecoin against their deposited assets. Read this to get a complete understanding of how they work.*

There are a lot of existing stablecoin protocols. The largest and most widely used is MakerDAO's protocol which mint's the native stablecoin DAI.

Developer Docs:
https://makerdao.com/en/whitepaper/
https://docs.makerdao.com/

Another more recent project is the Liquity protocol and their native stablecoin LUSD. They offer greater capital efficiency (you can mint more stablecoins against the same assets) by utilizing a novel liquidation mechanism.

Developer Docs:
https://docs.liquity.org/

Finally, there are more exotic flavors of algorithmic stablecoins, some which attempt to maintain dollar parity without collateral but rather various algorithmic monetary policies. These are much more dangerous and prone to failure due to the fact that they rely on external market dynamics and "soft" pegs. They often fail catastrophically if certain economic limits are breached.

Read more here:
https://cryptobriefing.com/algorithmic-stablecoin-crashes-50-devs-scramble-fix/
https://cointelegraph.com/news/iron-finance-bank-run-stings-investors-a-lesson-for-all-stablecoins

What is an Automated Market Maker AKA Decentralized Exchanges

I have 1 bitcoin. I want to swap my bitcoin for 1 bitcoin worth of ethereum. I could deposit my bitcoin on a centralized exchange (covered later) and swap it just like selling 1 Apple stock and buying Tesla stock on a traditional stock exchange. These exchanges follow an order book model. I post an order to sell X Bitcoin for Y Ethereum (or Z USDC, N Dogecoin, etc.) and a counterparty with a matching buy order for X Bitcoin will fill that order. This works great on high speed centralized systems but falls apart very quickly on decentralized networks where transactions can only get executed every ~15 seconds. Imagine trying to implement a system meant to match thousands of orders a second on a decentralized computing network only capable of executing ~150 transactions every 15 seconds and having each transaction cost ~$50-$300.

Obviously we need a new system to exchange assets given the computational limits of blockchains.

Read this:
https://www.coindesk.com/learn/2021/08/20/what-is-an-automated-market-maker/

A look at Uniswap V2. The OG AMM

Understand Everything:
https://uniswap.org/whitepaper.pdf

Developer Docs:
https://docs.uniswap.org/protocol/V2/concepts/protocol-overview/how-uniswap-works
https://github.com/Uniswap/v2-core

Uniswap V3

(Optional)

Uniswap V3 introduces significantly more complexity to the previous AMM paradigm with the aim of increasing capital efficiency and reducing price impact of large trades.

Understand Everything:
https://uniswap.org/whitepaper-v3.pdf

Developer Docs:
https://docs.uniswap.org/protocol/introduction
https://github.com/Uniswap/v3-core

What is Curve

(Optional)

Curve can be best summarized as Uniswap V2, with more hard math to reduce slippage. Curve is an AMM primarily meant to swap assets of similar value. Curve's most liquid pool is 3CRV which consists of USDC, USDT and DAI. Curve's pricing formula allows significantly better pricing for large swaps from one stablecoin to another with reduced price impact compared to using something like Uniswap.

Understand Everything:
https://curve.fi/files/stableswap-paper.pdf

https://curve.fi/files/crypto-pools-paper.pdf

Developer Docs:
https://curve.readthedocs.io/
https://github.com/curvefi/curve-contract

Decentralized Bank

A traditional bank allows you to deposit money and take out loans. Decentralized banks are just like that. I can deposit a variety of cryptocurrencies (Bitcoin, Ethereum, USDC, USDT, etc.) and earn interest on my deposits. The interest is typically algorithmically controlled (i.e., if there are a lot of people borrowing bitcoin, I'll earn a higher interest rate by depositing bitcoin compared to depositing USDC). Similarly, I can also take out loans against my deposits. If I deposit 10,000 USDC, (depending on the protocol parameters) I can take out a loan of $9000 worth of Ethereum.

AAVE

Intro:
https://decrypt.co/resources/what-is-aave-inside-the-defi-lending-protocol

Understand Everything:
https://github.com/aave/protocol-v2/blob/master/aave-v2-whitepaper.pdf

Developer Docs:
https://docs.aave.com/developers/
https://github.com/aave/aave-protocol

Compound

(Optional)
Compound is similar to AAVE. Read this if you want to see a slightly different implementation but much of the core concepts remain the same.

Understand Everything:
https://compound.finance/documents/Compound.Whitepaper.pdf
Developer Docs:
https://compound.finance/docs
https://github.com/compound-finance/compound-protocol

Liquity/Maker Dai

The lines between Decentralized Lending Banks/Protocols and Decentralized Stablecoins get a little blurry. Both systems fundamentally allow you to deposit assets and take out a loan against these assets. The key difference is decentralized stablecoin protocols have the ability to mint new stablecoins out of thin air while lending protocols require other depositors to deposit assets before you are able to borrow them.

Oracles

Blockchain is a closed off distributed computer. It has no access to the rest of the World Wide Web. This means you cannot natively make an api call, google something, or download files from a smart contract living on Ethereum. Most importantly, you cannot even generate a cryptographically secure random number within a smart contract. Thus the concept of Oracles were invented.

Introductions:
https://chain.link/education/blockchain-oracles

Developer Docs:
https://docs.chain.link/

Forks

The overwhelming majority of blockchain code is open-sourced. To clarify, every piece of smart contract code on the blockchain is publicly accessible. However, the code that exists on the blockchain is effectively a compiled binary and is hard to read and understand. Most projects often post the original precompiled source code on GitHub so users, developers and auditors can verify it's security and legitimacy. This naturally makes it quite trivial for someone to copy and paste existing protocols and smart contracts and deploy them as their own and that is exactly what happens. These copy and pasted versions of protocols are commonly referred to as Forks. Forks have varying degrees of modifications. They can copy and paste the exact code or they can create significant modifications that add new features or security improvements (and unintentional security flaws too).

You might ask why anyone would use a fork rather than an original protocol. Oftentimes, forks employ various strategies to bring users and liquidity to their protocol. The most common strategy is called a Vampire Attack in which the fork offers users better rewards, fee rates, etc. to incentivize them to migrate.

SushiSwap was a copy and paste fork of Uniswap V2 that brilliantly executed a vampire attack on Uniswap V2 and is one of the most successful forks to date.

Read more here:
https://www.gemini.com/cryptopedia/sushiswap-uniswap-vampire-attack
https://finematics.com/vampire-attack-sushiswap-explained/

Centralized Exchanges

Centralized Exchanges are venues in which users can trade assets through a corporation's internalized system. Think of it like a stock exchange but for crypto. That's literally all it is. Centralized Exchanges are notably different that Decentralized Exchanges for a number of reasons:

1. ***Almost all Centralized Exchanges now require you to KYC** (Know Your Customer) due to a tightening of regulatory requirements. This typically involves providing sensitive personal information like your full name, address, SSN, passport, etc. Centralized Exchanges may also block*

users in certain geographical areas due to regulatory requirements. (Binance blocks all US based users for example). Decentralized Exchanges have no such requirements.

2. ***Centralized Exchanges custody the assets you wish to trade with****. This means to trade, you have to send your cryptocurrency to the exchange's own wallet. This has a number of implications, most importantly security. If the Centralized Exchange gets hacked, you have no recourse. This is not like the stock market where you can always undo a transaction that was proven to be malicious/illegal/unauthorized. The blockchain does not care whether or not the transaction was unintentional. If the transaction was signed with the valid private key, it will be executed, accepted and immutably recorded on the blockchain. Hacks happen a lot:* *https://www.wired.com/2014/03/bitcoin-exchange/*
3. ***Centralized Exchanges have costs associated with withdrawing or depositing assets****. If you make a trade on Uniswap, immediately after the trade, your wallet will contain the assets you just purchased. In contrast, centralized exchanges have varying wait times if you wish to withdraw your assets after making a trade. In the absolute best case it'll take ~5min for you to initiate the withdrawal request, have the exchange process the request and send your assets to the indicated wallet. In the worst case (especially if you're doing a large transfer), exchanges may require additional verification, waiting periods for bank funds to settle (if you paid with fiat), or even delays in the event there is insufficient liquidity and the exchange has to unlock funds stored in cold wallets. Furthermore some exchanges may charge withdrawal and deposit fees in addition to the gas fees you pay for the on chain transfer.*

There are a lot of centralized exchanges with varying degrees of regulatory compliance, security and user experience. Some top exchanges by reputation and volume:
https://www.coinbase.com/
https://www.binance.com/en
https://www.kraken.com/
https://www.gemini.com/

MEV

Now that you've developed a basic understanding of some of the lego pieces in the blockchain ecosystem it's time to dive into the world of MEV. MEV, Maximal Extractable Value/Miner Extractable Value is basically any automated interaction you can do on the blockchain that has positive expected value. Importantly, it is ***not*** *speculating on the prices of assets, YOLOing into a ponzi-esque yield farming protocol or any sort of traditional investment strategy. One kind of accurate way to think about it: MEV is to DeFi what HFT is to the stock market.*

Introduction:
https://ethereum.org/en/developers/docs/mev/
https://research.paradigm.xyz/MEV

The OG Flashboys 2.0 (Hard. Required if you want to build something. Seriously. Read it first):
https://arxiv.org/pdf/1904.05234.pdf

Very Very Fun Reads (Not hard. Just insightful):
https://www.paradigm.xyz/2020/08/ethereum-is-a-dark-forest/

https://samczsun.com/escaping-the-dark-forest/
https://rekt.news/return-to-the-dark-forest/

MEV Strats 101

MEV strategies basically consist of a set of on-chain interactions with the goal of ending up with more money than you started with by the end of the execution sequence.

Sandwiching

Sandwiches are the most notorious form of MEV. They are the blockchain equivalent to the frontrunning strategies employed by HFT firms laid out in Michael Lewis's Flash Boys. At its core a sandwich exploits the existence of pending transactions waiting to be confirmed on the blockchain. If you see a pending order to buy 100,000 USDC worth of Ethereum through Uniswap, the logical conclusion is that the price of Ethereum will go up after that transaction has been executed. You want to buy Ethereum immediately before the targeted transaction has been executed and then immediately sell that same Ethereum right after the target transaction executed.

*In practice, you can use the pricing formulas detailed in the Uniswap whitepapers to calculate the *exact* price impact that any order will have on a trading pair. This is a crucial advantage that sandwich attacks have over tradfi frontrunning where slippage is a probabilistic calculation. The deterministic nature of each individual trade on decentralized exchanges allows you to derive and calculate optimal trade sizes to frontrun a purchase with, allowing you to extract the globally maximal profit from any sandwich attack.*

Learn more here:
https://medium.com/coinmonks/defi-sandwich-attack-explain-776f6f43b2fd

***A quick ethics tangent:** Sandwich attacks are typically excoriated within the DeFi space and viewed as negative externality. There are two ways to look at Sandwich Attacks. The Bad Way and the Good Way.*

***The Bad Way** views sandwich attacks as a purely exploitive strategy that preys upon uninformed retail users who don't fully understand the underlying network dynamics of blockchain and the implications of the slippage parameters they set when making a trade. The most brutal sandwich attacks often occur when a user YOLOs into a newly launched shitcoin and through a desire to be one of the first buyers of the coin set slippage to absurdly high values (sometimes as high as 99%) inevitably resulting in them receiving exactly their minimum specified output amount — this amount is an order of magnitude lower than they otherwise would have received in the absence of a sandwich attack.*

***The Good Way** views the entirety of this set of interactions through the lens of economic efficiency. When a user submits a swap, they effectively declare their maximum willingness to pay through the parameter minOutput/maxInput. If their order fills at a more favorable* *price than the declared WTP, it implies the existence of a consumer surplus and consequently the absence of a Nash Equilibrium. A third party, the sandwich bot operator, moves the economic system towards the Nash Equilibrium such that the market meets the consumer's maximum WTP while also extracting value for the operator. In doing so, the sandwich bot has generated more global utility and reached the Nash Equilibrium where no other action would produce additional utility given the current state of the system and the declared economic preferences of all agents involved.*

Both ways are reasonable positions to view sandwich attacks and in truth the reality likely lies somewhere along the spectrum of the Good Way and the Bad Way. A couple parting thoughts to ponder:

- *Sandwich attacks will always exist. If you don't run a bot, someone else will*
- *This practice is widespread in the equities market and an accepted dynamic*
    - *Frontrunning as referenced in Flashboys still occurs to this day*
    - *A novel trade mechanism called* [Payment for Order Flow (PFOF)] *has enabled retail brokerages to offer no-fee trades by routing orders through market makers like Citadel. These market makers pay brokerages for retail order flow since they don't face the same degree of adverse selection they otherwise would when trading against informed counterparties on the public stock exchange*
    - *The effects of HFT practices in equities are far more opaque due to the existence of Dark Pools, the centralized nature of equity markets and complexity of equity market architectures*

Arbitrage

Arbitrage involves buying an asset at a discount on one venue and simultaneously selling it at a different venue for a higher price. Arbitrage is a mechanism to make markets efficient and reduces price discrepancies of assets across different venues.

Read more here:
https://www.coindesk.com/learn/crypto-arbitrage-trading-how-to-make-low-risk-gains/

There are many many different flavors of arbitrage. Some are absolutely considered to be MEV while others incorporate more traditional financial strategies:

Atomic Arbitrage

If you paid attention to the smart contracts section you should know that smart contracts allow you to package a series of on chain interactions, sequentially execute them, and at the end of execution, check for a set of conditions. If the conditions are not met, the smart contract can force the execution to fail effectively undoing all the on chain interactions that just occurred.

The implications of this mechanism are enormous. In traditional forex arbitrage, when you execute a multi hop arbitrage, you run the risk that the market moves against your position while you are halfway through your arbitrage forcing you to sell at a loss to return to your *original denominated currency. With the existence of atomic execution, not only will every leg of your hop be guaranteed to execute one after another with no race conditions (Ethereum is a singular global state machine that can only process transactions sequentially), but your smart contract code can record your initial starting balance, and at the end of execution can assert startingBalance<endBalance. If the assertion fails, the smart contract will revert the execution, undoing all the trades made in the function call and leaving you with your starting balance.*

This ability to revert a smart contract function call essentially makes this arbitrage strategy riskless in theory. There are some caveats to this "riskless" assertion that will be covered later.

Furthermore, the existence of atomic execution brings about novel economic mechanisms that were impossible to implement in the traditional computing paradigm. Namely flashloans and flashswaps. DeFi protocols such as Uniswap and AAVE consist of smart contracts that hold billions of dollars in capital. Traditionally, loans had to be secured with collateral, otherwise the borrower could just run off with the money with little recourse for the lender. With atomic execution however, the borrower's smart contract could request a loan by calling a function in the protocol's smart contract. Within the same execution context, the protocol's smart contract will optimistically provide the loan to the borrower, then call a predefined function in the borrower's smart contract letting it know the loan has been provided. The borrower's predefined function will execute and at the end of execution return the loaned assets (plus a fee) back to the protocol's smart contract. The protocol's smart contract function that was initially called, providing the loan and calling the borrower's smart contract will execute a check when the borrower's function finishes execution that the loaned amount + fee has been returned. If the loan has not been returned, the check will fail and the entire chain of interactions (the borrowing contract's function executions & the initial transfer of the loan to the borrower) will revert/rollback to the initial state as if nothing had happened.

This concept of flashloans is arguably one of the hardest concepts to grasp in the space and requires a fundamental shift in understanding how programmable logic is executed on the blockchain compared to traditional computing.

It's crucial you understand this concept as it affords an unprecedented level of flexibility in many DeFi interactions. Let's tie this concept back to the atomic arbitrage strategy just described. Suppose you identify a triangular arbitrage opportunity that had a globally optimal input trade size of $1 million (recall that trades have price impact. If you execute an arbitrage with greater than the optimal amount, the price impact of your own trade will eat into the profits). If you only had $1,000 of capital available, you could only execute the arbitrage with your $1,000 — which might not even be profitable after paying for all the gas fees for your transaction to execute. With flashloans you could initiate your arbitrage by taking out a loan of $1 million, execute your triangular arbitrage, pay back the loan and keep all the arbitrage profits for yourself.

This mechanism allows for an incredible level of democratization in MEV strategies. Capital intensive strategies that were previously only accessible to those with access to large *amounts of capital are now accessible to individuals who could wield an almost unbounded amount of capital by sourcing it through DeFi protocols that provide flashloans.*

AAVE Flashloan:
https://docs.aave.com/faq/flash-loans

Uniswap Flashswap (Flashswaps allow you to borrow an asset and pay back the equivalent value denominated in a different currency which saves an extra hop):
https://docs.uniswap.org/protocol/V2/guides/smart-contract-integration/using-flash-swaps

Cross Centralized Exchange Arbitrage
Cross exchange arbitrage is closer to the traditionally employed forex arbitrage strategies. It involves making simultaneous buy and sell orders of the same asset across different exchanges both centralized and decentralized. Because centralized exchanges do not allow you to deposit, trade, and withdraw all in a single on chain transaction, atomic execution is not possible. Furthermore, the fees and delays associated with constantly depositing and withdrawing from centralized exchanges will inevitably force you to leave an inventory of assets across a variety of exchanges to trade against. If Ethereum is $3,900 on Coinbase but $4,000 on Binance, by the time you are able to withdraw the Ethereum you bought on Coinbase and deposit it into Binance, the arbitrage opportunity will be gone. Instead, you'll have to maintain a balance of Ethereum and Cash on both exchanges so you are able to immediately buy Ethereum on Coinbase and sell the Ethereum held on Binance without having to deposit or withdraw.

This forces you to engage in more traditional financial modeling and properly manage inventory risk of the assets you hold across all venues. This strategy is quite capital intensive due to the absence of flashloans.

A cool tool commonly used for this type of strategy:
https://hummingbot.io/

Hybrids
There exist a variety of Arbitrage strategies that utilize both on chain and centralized exchanges. Common hybrids involve integrating decentralized exchanges into the Cross Exchange Arbitrage and executing buy-sell arbitrage with inventory held in an on chain wallet. This is a strategy dominated by a specific searcher:
https://etherscan.io/address/0xa57bd00134b2850b2a1c55860c9e9ea100fdd6cf
You can observe just through the sheer volume of one sided buy/sell orders made on chain that this wallet is likely executing corresponding orders on other venues.

Another common strategy is to implement the cross exchange arbitrage across different blockchains. For example, a decentralized exchange on the Ethereum blockchain might have differing prices of assets than a decentralized exchange on the Polygon Network (an Ethereum-like blockchain).

Liquidations

DeFi Lending Protocols consist of a user borrowing assets against deposited collateral. What happens if the value of deposited collateral drops sharply? How do DeFi protocols ensure outstanding loans are always fully backed by collateral?

When a user takes out a loan, the loan is always overcollateralized (i.e., the value of collateral exceeds the value of the loan). However in the event the value of the collateral drops below a certain threshold (anywhere from 110% of the loan value to 300%), the DeFi protocol allows anyone to liquidate the loan position to maintain system solvency. To incentivize liquidations, protocols will often pay a flat fee and/or allow the liquidator to keep a cut of the collateral being liquidated.

Liquidations are always initiated by a smart contract function call to the DeFi protocol. The liquidator needs to repay some or all of the loaned amount at which point the protocol will transfer the borrower's deposited collateral to the liquidator (and sometimes an additional flat fee). The value of the deposited collateral should always be greater than the value of the loan (otherwise there would be no economic incentive to liquidate a loan).

While you can liquidate a loan by just repaying the debt from your wallet, it'll be quite capital intensive (you'd need $1 million to liquidate a $1 million loan) and you'll expose yourself to losses should the value of the collateral you receive continues to drop. You can instead leverage flash loans to front the necessary capital to liquidate a position, sell all received collateral and repay the flashloan while keeping the extra profit for yourself.

AAVE Liquidation Docs:
https://docs.aave.com/faq/liquidations

Generalized Frontrunning

When transactions are submitted to the blockchain, a node propagates the transaction to other nodes through a p2p gossiping mechanism. This means there will always be a non-zero amount of time between the time of submission and when the transaction is immutably confirmed on the blockchain. Now recall that Ethereum is effectively a publicly accessible state machine and each transaction represents a state change upon the global state. This means every single publicly broadcast unconfirmed transaction can be simulated upon the present state. A simple generalized frontrunner will simulate every unconfirmed transaction and observe the resulting state change. If it observes a net increase in balance by the initiating wallet of a certain transaction, it will attempt to replicate that exact transaction by copying all relevant transaction fields (where the transaction is being sent, data fields, gas values, etc.) and simulate its own replicated transaction on the global state. If it observes that it's own balance has increased after simulation, it will attempt to frontrun the original transaction by submitting its replicated transaction with a higher gas price.

In doing so, generalized frontrunners are able to capture arbitrary opportunities solely by mimicking the original sender's behavior and getting there first.

There is obviously more complexity to the generalized frontrunner active today. Advanced techniques include fuzzing of relevant transaction fields, even going as far as identifying and replacing mentions of the wallet address in call data or tracing smart contract interactions to identify specific subcalls within a transaction's execution that would result in a net balance increase.

Read the "Very Very Fun Reads" section to see some generalized frontrunners in action

Just In Time (JIT) Liquidity

Prerequisite: Read Uniswap V3

Uniswap V3 allows liquidity providers to specify a range of prices that their liquidity is active for. This allows for greater capital efficiency, if you expect the value of two tokens in a Uniswap V3 to stay relatively correlated, you can set a tighter range and receive a larger share of the transaction fees for swaps that occur within your specified range.

Depending on how tight you set the liquidity range, your share of fees may be many many orders of magnitude greater than setting a wider tick range. However, the tighter you set your tick range, the greater your risk of impermanence loss is, and the greater the likelihood that the price of the asset pair will move out of your tick range due to general volatility. When the price of the pair moves outside of your tick range, you generate no fees.

Now suppose you identify a large pending buy order of $10 million USDC for ETH on Uniswap V3. Let's say the fee rate for this pair is 0.3%. That means this order will generate $30,000 in transaction fees to be paid to liquidity providers. If you fortran this order and submitted a transaction to provide $10 million dollars of liquidity at the tightest possible price range, your liquidity position would receive a disproportionately greater share of fee revenue than all other liquidity providers who provide liquidity for a wider range of prices. You could then immediately remove liquidity after this order.

Through this, you're able to capture most of the fees generated by large orders without facing any of the risks that providing liquidity to AMMs incur over a longer time horizon. Obviously due to the non-atomic nature of this strategy, it's quite capital intensive.

Read more here:
https://twitter.com/ChainsightA/status/1457958811243778052?s=20

Long Tail

Sandwiches, Arbitrage and Liquidations make up the overwhelming majority of MEV extracted. Coupled with the fact that all the aforementioned strategies are widely known, they are incredibly competitive. If you are reading this, attempting to productionize any of these strategies will likely be a futile exercise as experienced searchers have already implemented hyper optimized versions of this strategy. Let me emphasize. You will always be playing catch up.

Long Tail MEV encompasses all other forms of MEV not described here. Long Tail MEV describes niche, arcane, undiscovered MEV often realized through interacting with lesser known protocols, event based strategies, or unorthodox economic mechanisms.

Searching for Long Tail MEV will likely be the most fruitful use of your time, but by its very nature will require significant exploration and understanding of the DeFi space.

One such example:
https://twitter.com/ChainsightA/status/1460824051010744327?s=20

Examples

Now that you have a good understanding on the basics of MEV, let's go over some examples:

An incredibly well written walkthrough of what the process from start to finish looks like
https://bertcmiller.com/2021/09/05/mev-synthetix.html

Productionized Liquidator Bot:
https://github.com/haydenshively/New-Bedford

Rari's Liquidator Bot:
https://github.com/Rari-Capital/fuse-liquidator-bot

Flashbot's Beginner Arbitrage Bot:
https://github.com/flashbots/simple-arbitrage

PGAs

Covered in Flashboys 2.0 mentioned above. Just read it.

Priority Gas Auctions occur when two or more searchers identify the same MEV opportunity and enter into a gas bidding war to capture the opportunity first. PGA's were commonplace in early MEV strategies on Ethereum leading significant amounts of spam and reverting transactions while congesting the overall network. The creation of Flashbots (covered next) eliminated the overwhelmingly majority of PGAs on Ethereum. However, PGA strategies have seen a resurgence with the deployment of many new EVM compatible chains that offer lower fees and lower block times.

Flashbots

What's Flashbots:
https://medium.com/flashbots/frontrunning-the-mev-crisis-40629a613752
Flashbots brought about a paradigm shift for MEV strategies on Ethereum. Most notably, multi-transaction atomicity and protection against reverts. Miners connected to Flashbots receive Flashbot bundles to execute. Searchers who identify MEV opportunities can submit Flashbot bundles to the relay. A Flashbot bundle is a list consisting of 1 or more Ethereum transactions. When a

searcher identifies an MEV opportunity, rather than submitting a transaction to be broadcast publicly through nodes, they submit the transaction to the Flashbot's relay. The Flashbot relay (after running through a series of checks) will then forward the transaction directly to the miner bypassing the p2p gossip process.

A couple key points

- *If a transaction submitted to the relay reverts/fails it will not land on chain. Flashbots ensures every bundle it receives is successfully executed on chain or not at all*
- *Bundles allow searchers to specify more granular transaction ordering preference. A bundle containing transactions A,B,C will execute A,B,C in the exact order provided. If any of the transactions revert, then none of A,B,C land on chain (unless reverts are explicitly allowed)*
- *Any valid transaction (properly formatted and signed) can be included in a bundle. Sandwich attacks will include the targeted transaction in their bundle ensuring the corresponding buy and sell transactions are place immediately before and after the targeted transaction*
- *Bundles sent to the relay indicate which block the bundle is valid for. This allows searchers to precisely target a specific block N+1, N+2, etc. to execute their transaction for*
- *If two bundles conflict (have the same transactions, cause one or the other to revert, etc.), the bundle with the highest effective gas price will execute.*

Learn Everything Here:
https://docs.flashbots.net/
https://github.com/flashbots

Smart Contracts / Dev Tooling

If you've gone through the flashbot docs properly, you'll quickly realize that gas efficiency is key to winning an opportunity. You cannot successfully win a sandwich attack by simply making your buy and sell swaps through the Uniswap Router contract. It is simply too gas intensive. Building a smart contract for MEV is notably different than building a smart contract for a Dapp. An MEV smart contract is stripped down to the bare minimum — prioritizing gas efficiency above all else (including UX, readability, and sometimes even security).

A few guidelines:

- *Any computation that can be done off chain must be done off chain*
- *Redundant variables need to be removed. Use things like msg.value, msg.sender for extra data and strip your calldata to the minimum*
- *Interact with the minimum amount of contracts possible. For example, don't swap through the Uniswap router, call the swap() function on the pair directly.*
- *NEVER use storage. Access variables through memory and calldata only. Storage reads and writes are excruciatingly expensive gas wise and are almost always unnecessary.*

See recent flashbot bundles:
https://flashbots-explorer.marto.lol/

Some active MEV Bots:
https://etherscan.io/accounts/label/mev-bot

MEV Job Board:
https://github.com/flashbots/mev-job-board

Decode and Analyze Transactions:
https://etherscan.io/
https://ethtx.info/

Web3 Libraries for your Backend:
https://github.com/ethereum/web3.py
https://github.com/ChainSafe/web3.js
https://github.com/gakonst/ethers-rs

Test Your Contracts:
Mainstream (prevalent in Dapps):
https://eth-brownie.readthedocs.io/en/stable/
https://hardhat.org/
More suited for searchers:
https://dapp.tools/
https://github.com/gakonst/foundry

Run an Ethereum Node:
Flashbot Version of the most commonly used ethereum client
https://github.com/flashbots/mev-geth
Erigon, a highly efficient archival client capable of storing the entire blockchain locally (also provides useful tracing functions):
https://github.com/ledgerwatch/erigon

Monitor your scripts/setup your cli quickly:
https://github.com/tmux/tmux
https://github.com/tmuxinator/tmuxinator

Game Theory / Social Dynamics / Security

This is the Flashbot's discord: https://discord.gg/7hvTycdNcK

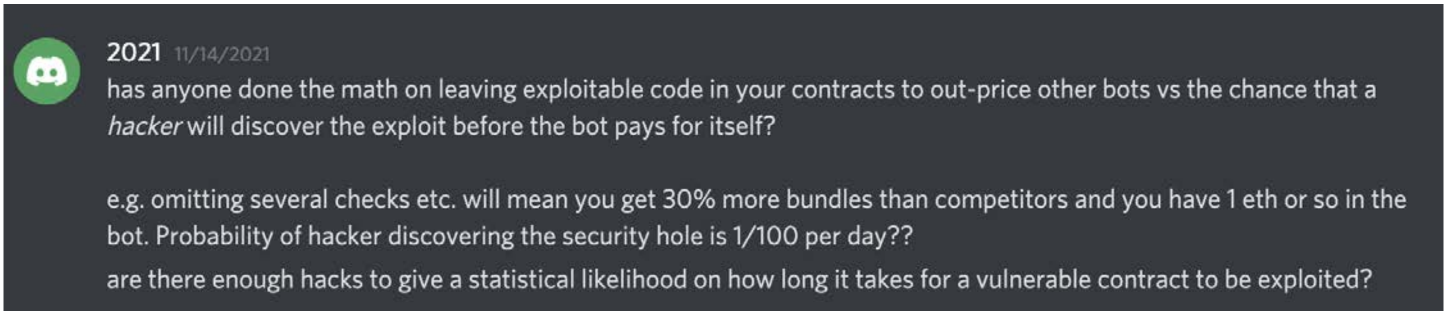
If you're serious about building something, join it and spend a day reading through past messages. I guarantee you a question you have will have been asked and (probably) answered in this discord.

The social dynamics among searchers are incredibly interesting and unique — a phenomena unlike any other ecosystem in the world. It's a contradictory mix of ruthless adversity and benevolent hand holding that really can't be found anywhere else.

Some basic tidbits:

- **Everyone and I mean *everyone* will be trying to steal your money**
    - I cannot emphasize this enough
    - If you write a vulnerable smart contract or run a vulnerable strategy and put money into it. It will get exploited and taken. It's not a question of if it's a matter of when

    - https://github.com/Defi-Cartel/salmonella
    - https://twitter.com/bertcmiller/status/1381296074086830091
    - https://twitter.com/bertcmiller/status/1446416008709918732
    - https://twitter.com/qiushui777/status/1450656352292925441
- When a searcher steals your money, sometimes they're nice and give some, most, or even all of it back
    - Yeah, it's weird. I know. Especially since most of the time they can take it with little consequence
    - When my smart contract got exploited in this attack, the searcher who executed the exploit reached out and gave some of it back. It's pretty common.
    - https://twitter.com/bertcmiller/status/1448063805695696902
- It's kinda ok to steal another searcher's money?????
    - The main overarching rationalization is vulnerabilities are often a result of insane gas optimizations
    - By stripping your code down and removing as much of the overhead that built in safeguards incur, you've made your bot an order of magnitude more competitive
    - If you leave your optimized contract vulnerable, you're essentially accepting that the risk of an exploit outweighs the competitive advantage of your newly optimized code

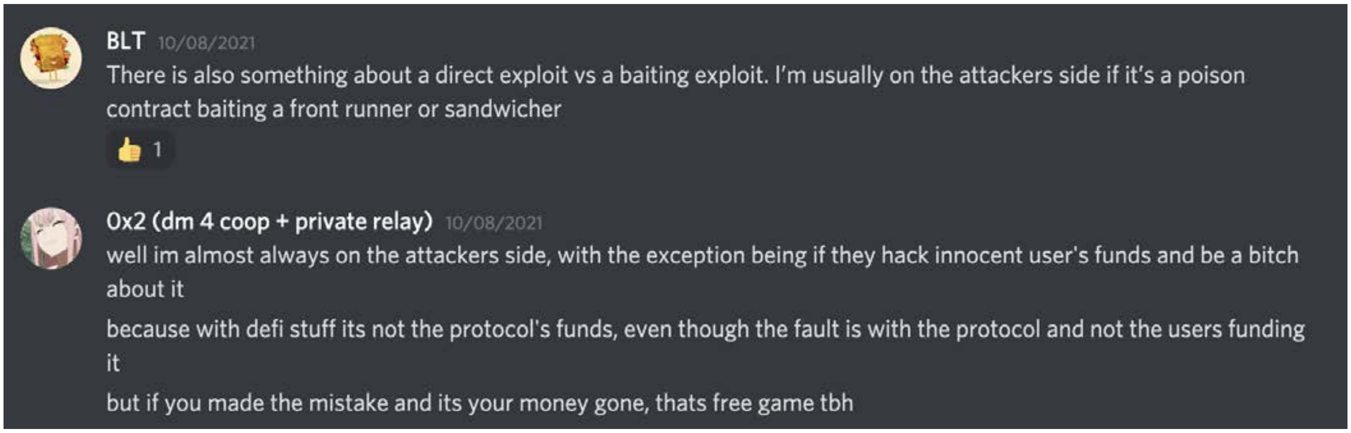


2021 11/14/2021

has anyone done the math on leaving exploitable code in your contracts to out-price other bots vs the chance that a *hacker* will discover the exploit before the bot pays for itself?

e.g. omitting several checks etc. will mean you get 30% more bundles than competitors and you have 1 eth or so in the bot. Probability of hacker discovering the security hole is 1/100 per day??

are there enough hacks to give a statistical likelihood on how long it takes for a vulnerable contract to be exploited?



BLT 10/08/2021

There is also something about a direct exploit vs a baiting exploit. I'm usually on the attackers side if it's a poison contract baiting a front runner or sandwicher

👍 1

0x2 (dm 4 coop + private relay) 10/08/2021

well im almost always on the attackers side, with the exception being if they hack innocent user's funds and be a bitch about it

because with defi stuff its not the protocol's funds, even though the fault is with the protocol and not the users funding it

but if you made the mistake and its your money gone, thats free game tbh

- Searchers throw tantrums when other searchers leak alpha
    - For hopefully obvious reasons
    - Most MEV is winner take all, there's really no room for sharing
- Crypto Twitter (specifically the MEV niche, not the degen ponzi scheme niche) is the best way to stay up to date on the latest developments. Some good accounts to follow: https://twitter.com/0xmebius/following
- Just a good read:

https://twitter.com/bertcmiller/status/1402665992422047747

Alpha Leaks

Now that you've learned most of what is publicly available about MEV let's share some secrets — stuff you either can't find elsewhere or are buried deep in arcane forums. Here are a few hacks and efficiency optimizations you can use to get a head start.

ERC-20 Hacks

- If you need to do ERC-20 transfers, just keep the balance in the smart contract and call transfer(). Trying to do an approve, transferFrom regimen with an EOA holding the coins is useless and isn't actually any more secure
- When your smart contract does an ERC-20 transfer, always leave 1 unit of the smallest denomination of the token in your smart contract balance. It'll save gas the next time you interact with the token

Uniswap Hacks

- When calling swap() for a multihop arbitrage, set the 'to' field to the address of the next Uniswap pair in your arbitrage hop. This avoids having to transfer the received tokens from your smart contract address to the next pair on every hop

Calldata Hacks

- Let's say you need the number 8458 passed into your smart contract. Rather than having an extra uint argument in your function parameter, send 8458 wei to the smart contract when calling it. You'll be able to access the value 8458 as the variable msg.value. Obviously this only works for numbers of small magnitude. You don't really want to have to send massive amounts ether to your contract but 8458 wei is an infinitesimal amount to send
- Pack your calldata. Normally, each argument in your calldata will be padded with leading 0s by the abiEncode function. This means you'll often end up with extra 0s in your calldata that do nothing other than pad. Extra bytes mean extra gas. If you get your hands a bit dirty using inline assembly, you can bypass solidity's standard abi decoder for calldata and just slice the calldata bytearray yourself:
https://gist.github.com/0xmebius/f161abff38b88c9005db31d47e39bbe0

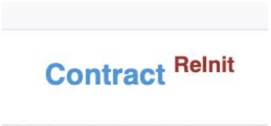
Leading 0s

- Ever wonder why a lot of MEV bots have a ton of leading 0s in the contract address? The TLDR is 0 byte calldata is marginally more gas efficient than non zero call data
- https://medium.com/coinmonks/on-efficient-ethereum-addresses-3fef0596e263
- https://ethereum.stackexchange.com/questions/101311/how-much-gas-do-0x00000000-addresses-and-0x00000000-methods-really-save
- Mine leading 0 EOAs and contract addresses:
https://github.com/johguse/profanity

- Another leading 0 hack. When you define a function foo(uint x), the function signature (the first 4 bytes of your transaction calldata used to select which function you are calling) will be the first 4 bytes of Keccak-256 hash of foo(uint x). You can brute force various function names so that Keccak256(FUNCTIONNAME(uint x))[:4] == 0x0000. This saves some gas since 0 byte calldata is slightly cheaper than non zero byte calldata https://github.com/fxfactorial/cheap-name-

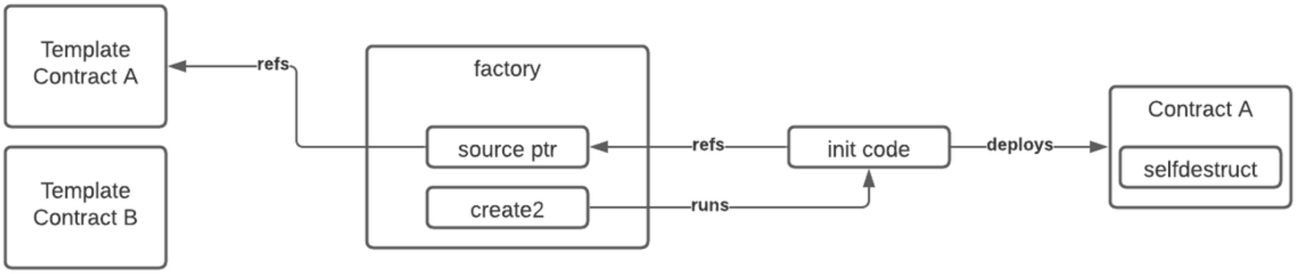
Ephemeral Smart Contracts

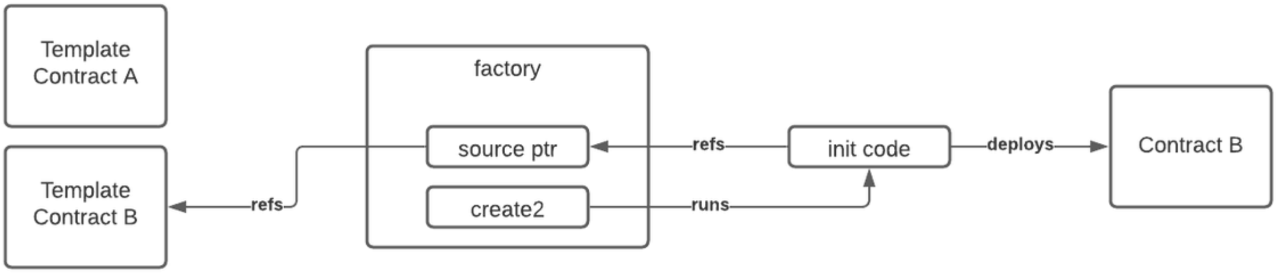
- Ever wonder why you see top MEV bots like this have the contract tagged with ReInit?



Contract ReInit

- What if I told you smart contracts, when deployed in a specific manner, are completely mutable? No proxies, no delegatecalls, just a pure editable smart contract
- Why would you want to edit an existing smart contract rather than redeploy?
  - If you already have a bunch of ERC-20 token balances, you can continue to benefit from the gas savings without having to reinitialize a bunch of storage slots
  - If you've mined a leading 0 smart contract address, redeploying means having to remine a new leading 0 address. This way, you can redeploy to the exact same address without having to spend weeks waiting for a new one
- By taking advantage of 2 low level operations, CREATE2 and SELFDESTRUCT, exposed in the EVM, you can deploy a smart contract, self destruct, and redeploy to the exact same address
- So how does this work? Best explained by libevm: https://twitter.com/libevm/status/1468390867996086275?s=20
- Here is one way you can setup the contract deployment (credit: @cupidhack):





- Here’s how you implement it:
  https://github.com/0age/metamorphic
  https://github.com/johguse/ERADICATE2

Get Information

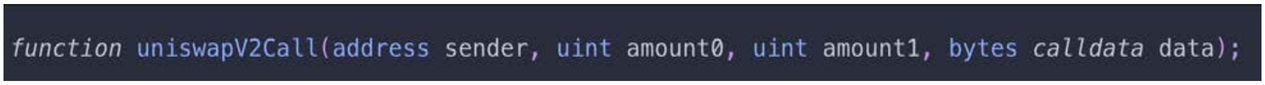
- Check the flashbots explorer every hour on the hour for event based MEV opportunities opportunities
- Batch on-chain queries using multicall contracts:
  https://github.com/indexed-finance/multicall
- If you need to access storage slots on chain, use geth’s graphql endpoint
  Inspired by @libevm: https://twitter.com/libevm/status/1467376978697211904

Flash Loans and Multicall

- You’ll reach a certain point where you realize that certain MEV strategies will require a lot of modularity
- For example, a general atomic arbitrage strategy might make anywhere from 2-N hops across different pairs. You could just write N-1 functions for each hop. But what if one of the pairs you’re hopping through has a slightly different function signature. Say swapToken() instead of swap()? Then you’ll have to write separate functions that call different function signatures and you’ll end up with a massive headache not to mention the gas costs associated with deploying a bloated contract.
- Wouldn’t it be easier if you could just parameterize every subcall you want to have executed. Kind of like a batch proxy? That’s what multicall contracts do:
  https://github.com/makerdao/multicall

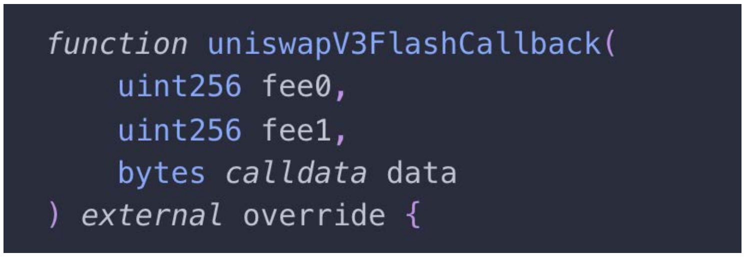
- This would be a good time to brush up on what delegatecall and call do: https://ethereum.stackexchange.com/questions/3667/difference-between-call-callcode-and-delegatecall
- Ok now that we can parameterize subcalls, how do we deal with flashloans? Recall that flashloans result in the loan provider calling back to your contract through a specific function signature.

For Uniswap V2 it's:



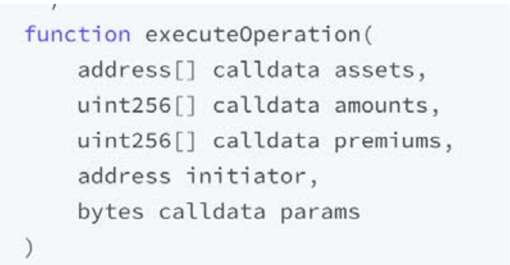
```
function uniswapV2Call(address sender, uint amount0, uint amount1, bytes calldata data);
```

For Uniswap V3 it's:



```
function uniswapV3FlashCallback(
    uint256 fee0,
    uint256 fee1,
    bytes calldata data
) external override {
```

For AAVE it's:



```
function executeOperation(
    address[] calldata assets,
    uint256[] calldata amounts,
    uint256[] calldata premiums,
    address initiator,
    bytes calldata params
)
```

Now you could just add 3 separate functions with the same logic, just different selectors in your multicall contract but this breaks the modularity of the smart contract. Let's revisit some of our solidity, specifically the fallback function. A fallback function is called whenever a smart contract is called and the selector doesn't match any of the existing function selectors. It's basically a catch all. Now you can receive generalized flashloans simply by putting all the flashloan logic in the fallback function. Be warned there are serious security vulnerabilities that will need to be patched to make this function safe to expose. This will be left as an exercise to the reader :)

LEARN RUST

Rust is a high performance language that is now gaining significant traction both in the MEV community as well as the general software development industry. Rust offers the speed and low level interactions afforded by C but without the additional development overhead of worrying as much about memory allocation and garbage collection issues. It's also used as the base language for smart contracts on newer, high performance chains like Solana, Polkadot, Avalanche, etc.

Read:
https://doc.rust-lang.org/book/

Learn By Example:
https://doc.rust-lang.org/rust-by-example/hello.html