[

image

1500×1280 78.7 KB

](https://collective.flashbots.net/uploads/default/original/2X/3/30b16a9646d8b3e02d76ff16abd695cba60c6410.png)

Team Information

- Team Name:

Discreet Valorizers

- Team Members:
- Name 1: @brock
- Name 2: @halo3mic
- Name 1: @brock
- Name 2: @halo3mic

Project Repository

- [GitHub Link

](https://github.com/zeroXbrock/dammit-bobby)

Project Goal

- Project Name:

Bottom of the Block Yack Runner

- Brief Description:

POC of the bottom of the block backrunner usingYakSwap on SUAVE.

- Motivation

In recent years, the Ethereum community has become increasingly aware of the negative externalities of harmful MEV practices like sandwich attacks. To mitigate these risks, many traders have turned to sending trade orders to block builders privately rather than exposing them in a public mempool where they might be exploited. This shift has significantly increased the prevalence of private order flow and its share in the blocks.

Some of these private orders can move the markets out of equilibrium enough to create arbitrage opportunities. However, due to their confidential nature only block builder is able to capitalise on them. Often, it proves to be more efficient for block builders to outsource this operation to an entity that specializes in arbitrage value extraction, but in doing so, the builder leaks some of the data that was meant to remain private and only accessible by the builder. Builders can minimize the leaked data by sharing only the difference in the final state of the built block with a select few searchers. Nevertheless, even with this approach, the potential consequences of leaking the state must be considered case-by-case.

An alternative approach that has emerged with SUAVE is to allow credible programs to search for arbitrage opportunities in a confidential execution environment, eliminating the need to leak information to untrusted parties. This method ensures that sensitive data remains secure while still enabling the identification of profitable arbitrage opportunities.

- Idea

The most efficient way to implement back-of-the-block backrunner is to allow existing searchers deploy their bots in their preferred language and configuration. However, the main challenge with this approach lies in ensuring the credibility of their computation. To prevent leakage of order flow, the bot would need to be deployed in a sandboxed environment inside a trusted enclave. While this setup presents a great future direction, such a setup is not yet possible.

An alternative approach would be to implement the searching functionality in Solidity and execute it inside MEVM. Fortunately, YakSwap, an on-chain Solidity aggregator, already provides such a solution.

While primarily designed and mainly used to act as an aggregator finding optimal paths between two assets, it can also find and execute arbitrage opportunities. Such use case has been observed in the wild, with one example transaction showing a fortunate trader extracting $4000 in profit.

- Goals

End-to-end test that showcases the bottom of the block backrun using YakSwap.

Challenges

- What were the major challenges your team encountered during the hackathon?
- Lack of dexes on Holesky:

SUAVE supports Holesky for block building, so we decided to use it for POC. The issue we encountered was that we couldn't find any existing DEXs on Holesky and had to deploy them ourselves. As we didn't want to spend the whole hackathon deploying dexes, we resorted to deploying solely three UniswapV2 pools across which we simulated arbitrage opportunities.

- No builder session call support:

The builder session is a relatively new feature in suave-execution-geth and didn't support a call to the state with applied transactions. We implemented this functionality ourselves on suave-execution-geth and suave-std.

- Building blocks from Forge:

Currently, to build blocks from MEVM, object BuildBlockArgs

needs to be sent to the suave-execution-geth node. Some of the object's arguments can only be obtained by subscribing to events emitted by a private beacon node. This complicates obtaining the necessary arguments for building a block from Forge. An approach to mitigate this issue is to capture these arguments inside suave-geth/suave-execution-geth instead.

- Lack of dexes on Holesky:

SUAVE supports Holesky for block building, so we decided to use it for POC. The issue we encountered was that we couldn't find any existing DEXs on Holesky and had to deploy them ourselves. As we didn't want to spend the whole hackathon deploying dexes, we resorted to deploying solely three UniswapV2 pools across which we simulated arbitrage opportunities.

- No builder session call support:

The builder session is a relatively new feature in suave-execution-geth and didn't support a call to the state with applied transactions. We implemented this functionality ourselves on suave-execution-geth and suave-std.

- Building blocks from Forge:

Currently, to build blocks from MEVM, object BuildBlockArgs

needs to be sent to the suave-execution-geth node. Some of the object's arguments can only be obtained by subscribing to events emitted by a private beacon node. This complicates obtaining the necessary arguments for building a block from Forge. An approach to mitigate this issue is to capture these arguments inside suave-geth/suave-execution-geth instead.

Current State

- Project Status at the End of Hackathon:

We implemented end-to-end test showcasing how YakSwap can be leveraged to find and execute bottom-of-the-block backrun for blocks built on SUAVE. 1. The demo begins with YakSwap being used to find top-of-the-block arbitrage;

1. Then a block is partially built by adding a single trade to it that makes DEX pools unbalanced and creates an arbitrage opportunity;
2. YakSwap is used again to find arbitrage opportunities at the bottom of the partially built block;
3. The transaction extracting found arbitrage opportunity is added to the bottom of the block;
4. The bottom of the partially built block is searched again for arbitrage, showing that the last transaction cleared some of the profits.
5. The demo begins with YakSwap being used to find top-of-the-block arbitrage;
6. Then a block is partially built by adding a single trade to it that makes DEX pools unbalanced and creates an arbitrage opportunity;
7. YakSwap is used again to find arbitrage opportunities at the bottom of the partially built block;
8. The transaction extracting found arbitrage opportunity is added to the bottom of the block;

9. The bottom of the partially built block is searched again for arbitrage, showing that the last transaction cleared some of the profits.

[

image

2000×1304 215 KB

](https://collective.flashbots.net/uploads/default/original/2X/c/cab8707b55f6050dc6397b6c69e4355a31f163d0.jpeg)

- What is still a work in progress?
- The optimal amount:

The current implementation only checks arbitrage opportunities for a single input amount. Such an approach is unlikely to capture the majority of arbitrage opportunities, and some search for the optimal input amount needs to be implemented. A golden section search is an efficient way to find the optimal amount.

- Proposing a block:

The present implementation only fills the block with the transactions, but it doesn't build it and submit it to the relay. While building a block from the filled transaction is trivial submitting a valid block to the relay involves obtaining certain beacon chain arguments that are challenging to fetch from Forge environment. See "Building blocks from Forge" from challenges section for more information.

- End2End test reliability:

The UniswapV2 pairs used in the test are deployed on Holesky, and their reserves can be updated outside the test environment, thus introducing inconsistent arbitrage opportunities between different test runs. Note that currently, we control the tokens associated with the pair, and the test doesn't apply the arbitrage transaction to the live Holesky chain, thus not changing the reserves—this might not hold in the future.

One approach would be to use suavex-anvil to fork L1 at a given block.

- Confidential storage:

For now, the private key used for testing is exposed publicly, but in the future, confidential storage should be used.

- Latency considerations for a realistic scenario:

In the implemented end-to-end test, YakSwap searches for arbitrage only between three UniswapV2 pools, which doesn't consume much resources. For real-world use cases, there are many more pools to search through, and the DEX logic for them is more complex than for UniswapV2. It is unknown what the latency for searching in an environment as complex as Ethereum would be.

Research should be conducted to test the search latency for more complex scenarios to determine the feasibility of using YakSwap for the bottom-of-the-block backrunning on Ethereum.

Note that YakSwap is deployed on Arbitrum and Avalanche, supporting the majority of the DEXs deployed there. Tests on Arbitrum and Avalanche could give a good glimpse into the latency without the need to deploy YakSwap on Ethereum.

- The optimal amount:

The current implementation only checks arbitrage opportunities for a single input amount. Such an approach is unlikely to capture the majority of arbitrage opportunities, and some search for the optimal input amount needs to be implemented. A golden section search is an efficient way to find the optimal amount.

- Proposing a block:

The present implementation only fills the block with the transactions, but it doesn't build it and submit it to the relay. While building a block from the filled transaction is trivial submitting a valid block to the relay involves obtaining certain beacon chain arguments that are challenging to fetch from Forge environment. See "Building blocks from Forge" from challenges section for more information.

- End2End test reliability:

The UniswapV2 pairs used in the test are deployed on Holesky, and their reserves can be updated outside the test environment, thus introducing inconsistent arbitrage opportunities between different test runs. Note that currently, we control the tokens associated with the pair, and the test doesn't apply the arbitrage transaction to the live Holesky chain, thus not changing the reserves—this might not hold in the future.

One approach would be to use suavex-anvil to fork L1 at a given block.

- Confidential storage:

For now, the private key used for testing is exposed publicly, but in the future, confidential storage should be used.

- Latency considerations for a realistic scenario:

In the implemented end-to-end test, YakSwap searches for arbitrage only between three UniswapV2 pools, which doesn't consume much resources. For real-world use cases, there are many more pools to search through, and the DEX logic for them is more complex than for UniswapV2. It is unknown what the latency for searching in an environment as complex as Ethereum would be.

Research should be conducted to test the search latency for more complex scenarios to determine the feasibility of using YakSwap for the bottom-of-the-block backrunning on Ethereum.

Note that YakSwap is deployed on Arbitrum and Avalanche, supporting the majority of the DEXs deployed there. Tests on Arbitrum and Avalanche could give a good glimpse into the latency without the need to deploy YakSwap on Ethereum.

- Are there any known bugs or issues?
- See "End2End test reliability" from the previous question
- See "End2End test reliability" from the previous question

How to Run

See README.md.

Additional Notes

- Any other information:
- Future plans for the project
- Arbitrage with optimal amount:

Implementing golden section search to find the optimal amount

- Block production:

Publishing the backrunned block to Holesky

- Latency considerations:

Research the latency considerations of complex DEX scenario

- End-to-end test improvements
- Reliability
- Usage of confidential storage
- Reliability
- Usage of confidential storage
- Arbitrage with optimal amount:

Implementing golden section search to find the optimal amount

- Block production:

Publishing the backrunned block to Holesky

- Latency considerations:

Research the latency considerations of complex DEX scenario

- End-to-end test improvements
- Reliability

- Usage of confidential storage
- Reliability
- Usage of confidential storage
- Acknowledgments
- This project would not have been possible without @ferranbt's help.
- The idea stems from the discussion: Request for SUAPP: SUAVE On-Chain Searcher Bot.
- This project would not have been possible without @ferranbt's help.
- The idea stems from the discussion: Request for SUAPP: SUAVE On-Chain Searcher Bot.
- Future plans for the project
- Arbitrage with optimal amount:

Implementing golden section search to find the optimal amount

- Block production:

Publishing the backrunned block to Holesky

- Latency considerations:

Research the latency considerations of complex DEX scenario

- End-to-end test improvements
- Reliability
- Usage of confidential storage
- Reliability
- Usage of confidential storage
- Arbitrage with optimal amount:

Implementing golden section search to find the optimal amount

- Block production:

Publishing the backrunned block to Holesky

- Latency considerations:

Research the latency considerations of complex DEX scenario

- End-to-end test improvements
- Reliability
- Usage of confidential storage
- Reliability
- Usage of confidential storage
- Acknowledgments
- This project would not have been possible without @ferranbt's help.
- The idea stems from the discussion: Request for SUAPP: SUAVE On-Chain Searcher Bot.
- This project would not have been possible without @ferranbt's help.
- The idea stems from the discussion: Request for SUAPP: SUAVE On-Chain Searcher Bot.