“With SUAVE I know that my orderflow can only be operated on safely via protecc’ed execution”

h/t to The Cybernetic Rooster for the framing

The Game

We have sensitive transactions T_i

collected from different users i

, these can be swaps or any type of transaction. We want a block builder with transaction ordering strategy S

represented as EVM bytecode to be able to run an execution like B := EvalStrategy(strategy=S, txns={T})

on a remote kettle, and we want to make sure S

does not:

- leak information
- break integrity on the underlying data

This game can be considered an extension ofprivate back running via MPC where EvalStrategy is a general interpreter and the private strategy S is block builder code.

At the limit, you could imagine Flashbots or other private transaction and bundle RPC providers pumping their RPCs through SUAVE, immediately making it available to any strategy S

, which adheres to some predefined rules, deployed to a remote kettle. Although an idealistic goal, this would “front-run” the current meta where block builders are bulk purchasing orderflow from other builders and allow anyone to build blocks on Flashbots + others flow.

Leaking Data

There are roughly a few side channels for leaking information from code running inside of the MEVM on a TEE:

1. memory access patterns
2. network calls
3. confidential storage
4. onchain storage
5. vulnerabilities to code

(1) can be handled by developing S

such that it can be run in an oblivious mode, but in general is a lower priority as our top concern is not leaking user orderflow over leaking a builder’s strategy.

(2) (3) and (4) can only be blocked via a modifier on the contract that the MEVM will check before giving the user orderflow. In fact, the MEVM already has a proof of conceptof such functionality when validating whether a precompile and associated callers are authorized to access data in the confidential data store.

Breaking integrity on the underlying data

SGX does a lot of the work here, but the next challenge is that the arbitrary strategy code S can contain within itscode for signing transactions in solidity, and thus, arbitrary strategy S

could sandwich without restrictions, which would be very bad!

One way we could get around this is by creating something like an orderflow merkle trie, where there is an evaluator that the produced block B must go through before being sent off, and this evaluator checks the inclusion proof of all transactions in the original merkle root. This comes with the downside that it limits the speed of incorporation of orderflow into the system, but it’s potentially a worthwhile idea for open access to orderflow for block-building algorithms.

[

Screenshot 2024-02-27 at 1.49.33 PM

2594×1454 256 KB

](https://collective.flashbots.net/uploads/default/original/2X/5/542478ea17e48a68987401179ce64385e465f1f7.png)