One important thing to realize is that secure enclaves are not a silver bullet. Building secure protocols around them, even if you assume the enclave itself is impenetrable, is challenging.

First, a quick description about the threat model at hand we use with secure enclaves in the Secret Network:

1. We assume each node is untrusted - i.e., could be run by a malicious host
2. We assume each node is equipped with a secure enclave (SGX in our instance), that can execute code/data in a trusted manner - i.e., its data can't be observed or manipulated by the host.
3. We further assume there's consensus in the network, which is another layer of security (explained below).
4. We also assume the usual PKI/cryptographic primitives are secure (signatures, encryption, etc…)

So what does this threat model give us (and what doesn't it)? At a first glance, it would seem assumption #2

gives us all we need - secret contracts run inside of enclaves and no one can see the code/data (privacy!) and and no one can tamper with the computations (correctness/integrity!). But it turns out this is only part of the picture.

Because of assumption #1

(host can be malicious), they can do whatever they want to the inputs going into the enclave/outputs coming from the enclave. Because of assumption #3

(consensus), attacks trying to tamper with the data (e.g., replacing a transaction or even censoring it) won't work, which is great!

However, this does leave out attacks where an untrusted host (from assumption #1

) tries to fool his own, local, secure enclave (assumption #2

), in order to leak data. Let's give a simple example:

Say there's a secret contract that acts as a time vault for secrets: users can store secrets, only to be released at a given time (say, a given block height).

Although from a consensus-perspective, the block height is provided and agreed upon by the network, which means you can't fool the network as a whole that we're at a different point in time than we really are, you can however, trick your OWN secure enclave by changing the current block height number and passing that instead to the enclave. Block height, like many other environment variables, are not authenticated in any way, and the enclave has no way to know if it can trust that input. Therefore, in this instance, it would be trivial for a malicious host to trick its enclave into thinking we are far enough in the future, so it makes sense to output the decrypted secret.

This, of course, is just one example of a class of problems - any data that comes into the enclave and is used by the enclave, cannot be trusted and needs to be authenticated. In many cases, this is easy, because the user herself shares the inputs and signs them, so a malicious host cannot manipulate that data. However, for other types of information, like meta-data that lives in the network (e.g., block height), there's no easy solution. This is also documented in this issue: Document limitations of information provided to secret contracts · Issue #201 · scrtlabs/SecretNetwork · GitHub.

We welcome feedback and ideas on how to resolve this concern. We have some ideas, but they aren't trivial and are too high level and unsubstantiated at this point. In the mean time, contract developers would need to be aware of this, and design their contracts accordingly to avoid trusting unauthenticated data from outside of the enclave.

- This problem could be seen as a tangent problem to the Oracle problem that exists in all blockchains.