Replay/side-chain

Transaction Replay Attacks/Side-chain attacks

After a contract runs on the chain, an attacker can sync up a node up to a specific block in the chain, and then call into the enclave with the same authenticated user inputs given to the enclave on-chain, but out-of-order, or omit selected messages. A contract not anticipating or protecting against this might end up de-anonymizing the information provided by users. For example, in a naive voting contract (or other personal data collection algorithm), we can de-anonymize a voter by re-running the vote without the target's request, and analyze the difference in final results.

Running complete sidechain attacks requires replaying the entire state and is not trivial and is mitigated completely if contacts have no Asynchronous inputs or outputs. Examples of affected contracts are mixers, here every user puts a TX in the pool for the final combination of TXs to be executed at once. An adversary could fill the pool with their own TX in a sidechain after every user input and deduct the user input from the retrieved output by eliminating the TXs they put in themselves.

For example the revoke message on Permits and regenerate message on viewing keys are subject to side-chain attacks as someone can create a forked chain where the revoke message never happens and retain access to the storage into the future.

One canno longer leverage tx simulation (after light-client implementation in v1.7) to perform non-fork replay attacks as described in apaper published in 2023 by Andrew Miller

Trusted and untrusted data - enclave light client

To prevent against Replay attacks and Simulation Secret implemented an enclave light client that verifies block commitments in V1.7.

In general, the light client protects Secret against fork attacks. However, it is important to know what exactly the light client checks to understand what the remaining attack surface is. The light client validates compute transactions and their order. Specifically, it matches the set of transactions to be executed with the data hash field of the block at the current height. However, the light client does not validate transactions from other modules (such as bank). The light client also does not validate internal contract calls (i.e. if your contract was called from another contract) as these are dynamic (cannot be predicted nor validated in advance) however, these are signed in a way that can only be generated by another contract inside the enclave.

This means that we can trust contract executions - User address, inputs, sent funds are all part of the signed and validated inputs.

State modification attacks

Looking at state modification attacks - these refer to attacks that attempt to modify the internal state of the chain to create behaviour that allows to compromise confidential data. This can be done by changing the internal state values of the chain (even encrypted values can be replaced with other valid encrypted values) and executing the real transactions to see what happens.

For example, imagine a contract that has the following logic:

1. Receive transaction from user with some amount of SCRT
2. Validate that payment is exactly 1000 SCRT
3. If payment is correct, reveal secret value
4.

An attacker can use a modified chain to attack this logic in the following way:

1. Send a transaction on the real chain with 1000 SCRT from an address that has 1 SCRT
2. Wait for the transaction to fail on mainnet
3. On a modified chain fund the attacker's address with 1000 SCRT
4. Replay the real block from mainnet
5. Reveal the secret value
6.

This scenario illustrates how payment systems need to consider these attack values.

Protecting against modified state is coming in Secret v1.11, and will be solved by validating that values that are read from the chain are part of the merkle tree of the chain itself.

Combating Replay attacks — The Millionaire's Problem

This attack provides a specific example of a tx replay attack extracting the full information of a client based on replaying a tx.

Specifically, assume for millionaire's that you have a contract where one person inputs their amount of money, then the other person does, then the contract sends them both a single bit saying who has more — this is the simplest implementation for Millionaire's problem-solving. As person 2, binary search the interval of possible money amounts person 1 could have — say you know person 1 has less than N dollars. First, query with N/2 as your value with your node detached from the wider network, get the single bit out (whether the true value is higher or lower), then repeat by re-syncing your node and calling in. By properties of binary search, in log(n) tries (where n is the size of the interval) you'll have the exact value of person 1's input.

The naive solution to this is requiring the node to successfully broadcast the data of person 1 and person 2 to the network before revealing an answer (which is an implicit heartbeat test, that also ensures the transaction isn't replay-able), but even that's imperfect since you can reload the contract and replay the network state up to that broadcast, restoring the original state of the contract, then perform the attack with repeated rollbacks.

You could maybe implement the contract with the help of a 3rd party. I.e. the 2 players send their amounts. When the 3rd party sends an approval tx only then the 2 players can query the result. However, this is not good UX.