Post co-authored with @lsankar4033

(note: 0x03 picked to disambiguate from WIP 0x02 credential PR)

Context

:

- Currently, in order for a validator to be ejected, they need to manually trigger it via a voluntary exit
- However, in cases such as staking pools, the owner of the funds being staked is not necessarily the validator. The validator has an opportunity to hold the owner's funds hostage
- We could get around this if there was a way for a smart contract to trigger voluntary exits and know that an exit has happened
- This proposal outlines a way to implement this:
- 'Exit' contract whose events are read by eth2 clients (similar to the Deposit Contract)
- Exposing the beacon state root to the eth1 engine so that the EVM knows whether a validator has been activated yet (by providing a merkle proof)
- 'Exit' contract whose events are read by eth2 clients (similar to the Deposit Contract)
- Exposing the beacon state root to the eth1 engine so that the EVM knows whether a validator has been activated yet (by providing a merkle proof)

Approach

We designate a canonical contract in eth1, the ExitContract, that looks like this gist.

Anyone can call the withdrawal method with a message originating from the specified credentials and a validator pubkey.

The Beacon chain clients listen for events from this contract and executes the following actions:

1. Looks up the withdrawal credential corresponding to the specified validator pubkey
2. Checks if:
3. the credential matches the one emitted by the event
4. The specified validator is active
5. the credential matches the one emitted by the event
6. The specified validator is active
7. If it does not, the algorithm terminates and is effectively a no-op.
8. Initiate the logic for a VoluntaryExit for the validator

In an ideal world, we'd also like the Ethereum chain to be able to assess whether a validator is active or not. This would allow smart contracts interacting with the ExitContract to provide additional 'safety checks' to their users, so that they can avoid getting in the no-op case by accident.

Strawman Example:

1. User burns 32 stETH expecting to reclaim 32 ETH once the corresponding validator is exited
2. The calling contract issues a message to the ExitContract specifying (by accident) a validator that isn't active yet
3. The emitted event does not cause the VoluntaryExit, since it falls in the 2b) case mentioned above, and gets interpreted as a no-op
4. The user does not receive any ETH since no validator is exited

To combat that, we could expose a BEACONROOT opcode at the EVM which would operate similar to the BLOCKHASH code. This would allow consumers of the BEACONROOT to be able to prove that a validator is active in the Beacon State. In the above example, the user would also provide a merkle proof showing that the chosen validator is active at the time the exit request is processed.

Next Steps

- Enshrining an ExitContract in Eth1 similar to the DepositContract
- Implementing the event processing logic in Eth2 clients, similarly to how they process validator entry events via the DepositContract
- Adding the BEACONROOT opcode at the EVM level
- Investigate the above protocol's soundness / DoS vectors etc.

We acknowledge that these are changes which SHOULD NOT be prioritized before The Merge. But we also believe it's worth having the discussion from now, so we can prioritize accordingly for when we'll enable withdrawals after The Merge.