Post co-authored with @vshvsh

This post is a follow-up of 0x03-withdrawal-credentials post and the discussions.

We use “withdrawer exit” throughout the text to name validator exit triggered by withdrawal credentials. We don’t consider 0x00 type withdrawal credentials, because to withdraw there should be a rotation to 0x01 type anyway.

This post aims to:

- elaborate on a possible solution for withdrawer exit requests
- present Generalized Message Bus (GMB) for conveying messages from the execution layer (EL) to the consensus layer (CL)

Motivation

Currently, control over withdrawal credentials doesn’t provide the capability to initiate the validator exit and withdraw the funds. Thus, when the withdrawals are enabled, delegated staking solutions will not be fully capable of fulfilling stakers requests to withdraw their funds. Thus it seems important that withdrawer exits get enabled in the Ethereum protocol along with the withdrawals.

Solution

Emit an event from a dedicated smart contract and process it on a beacon chain client similar to the DepositContract

mechanism. It’s quite alike the solution in 0x03-withdrawal-credentials-simple-eth1-triggerable-withdrawals but with the two main differences:

1. do no checks on EL which require knowledge of the Beacon state
2. generalize the pattern of conveying a message from EL to CL

The purpose of (1) is to avoid complexity and CL and EL coupling (if possible).

Generalizing

We think there is a pattern general and repetitive enough to consider it as a separate entity. The main components of the pattern are:

- need to convey a message from EL to CL
- need to protect CL from spam from EL

There are at least two types of messages to be sent from EL to CL: withdrawer exit and 0x01 type withdrawal credential rotation. Potentially, other new types are needed as the protocol evolves.

Full GMB spec is available here. Here we describe only the parts related to withdrawer exits.

On CL, we propose to implement a mapping from addresses of trusted contracts to lists of the events types expected from the address. A message is emitted on EL as an EVM event from a smart contract from an allow-list of message bus smart contracts. Then parsed, and processed on a beacon chain client like DepositEvent

from DepositContract

is processed (some related parts of the Prysm client: one, two, three).

Message of a GMB is a data structure Message

.

contract IBeaconChainMessageBus { struct Message { uint256 messageType; bytes data; }

event MessageEvent(Message message);

}

Depending on messageType

, data is parsed to a specific payload for that type.

As a basic way to protect CL from spam tx fees plus additional ETH burning are proposed to be used. If needed more specific checks can be implemented in the message bus smart contract.

WithdrawerExitBus contract

Basic (with no specific spam protection beyond EL transaction fees) WithdrawerExitBus

contract implementation based on GMB might look like this.

contract WithdrawerExitBus is IBeaconChainMessageBus { uint256 constant WITHDRAWER_EXIT_TYPE = 0x00;

```
struct WithdrawerExit {
    address withdrawalAddress;  // address corresponding to the 0x01 withdrawal creds
    uint256 validatorIndex; // caller is responsible to get it from validator pubkey
}

function initiateValidatorExit(uint256 _validatorIndex) external {
    // optionally require msg.value > some value and lock it on the contract
    WithdrawerExit memory withdrawerExit = WithdrawerExit(msg.sender, _validatorIndex);
    bytes memory encodedData = abi.encode(withdrawerExit);
    emit MessageEvent(Message(WITHDRAWER_EXIT_TYPE, encodedData));
}

}
```

We suppose it might be possible to provide a sufficient level of spam protection purely by gas costs + additional ETH burning.

The amount of non-malicious withdrawer exit requests is low and won't clog the queue. The risks come from invalid requests. The idea is to weed them out on CL, because the validity checks are relatively cheap and might not need the limit as low as we need it for voluntary exits.

Sufficient withdrawer exit request ETH costs might be derived from these limitations:

- non-malicious user costs (should be kept low when there's no attack)
- max amount of validity checks per block on CL (needs to be determined)
- tolerable duration of the withdrawer exit queue lock & price of the attack (needs to be determined)

For example, 24 hours blocking at 25 gwei gas price, and 24332 gas cost of a withdrawer request (call to initiateValidatorExit

) will cost:

max validity checks per block

block capacity filled

min attack price

16

3%

70 ETH

64

10%

280 ETH

615

100%

2693 ETH

Here is the python code to play with the parameters.

Block capacity filled is calculated for a block gas limit 15m. If it's 100% it means there is no space left in the block to cram more invalid withdrawer exits.

Note that the attack prices for high block capacity fills are more like lower limits because in these cases the gas price will go

up due to the base fee gas price increment.

In the extreme case, for 100% block capacity filled the gas price will become huge 31554 = 25 * 1.25**32

just after a single epoche (assuming the start base fee is 25 gas).

Consensus layer

A naive extension of the Beacon spec might look like this.

MAX_WITHDRAWER_EXITS = 2**4

class WithdrawerExit(Container): validator_index: ValidatorIndex withdrawer_address: ExecutionAddress

class BeaconBlockBody(Container): # ... deposits: List[Deposit, MAX_DEPOSITS] voluntary_exits: List[SignedVoluntaryExit, MAX_VOLUNTARY_EXITS] withdrawer_exits: List[WithdrawerExit, MAX_WITHDRAWER_EXITS]

def process_operations(state: BeaconState, body: BeaconBlockBody) -> None: # ... for_ops(body.deposits, process_deposit) for_ops(body.voluntary_exits, process_voluntary_exit) for_ops(body.withdrawer_exits, process_withdrawer_exit)

def is_withdrawer_exit_valid(state: BeaconState, withdrawer_exit: WithdrawerExit) -> bool: validator = state.validators[withdrawer_exit.validator_index] # Verify the validator is active assert is_active_validator(validator, get_current_epoch(state)) # Verify exit has not been initiated assert validator.exit_epoch == FAR_FUTURE_EPOCH # Verify the validator has been active long enough assert get_current_epoch(state) >= validator.activation_epoch + SHARD_COMMITTEE_PERIOD

Check withdrawer_exit was indeed initiated by withdrawal credentials
return withdrawer_exit.withdrawer_address == validator.withdrawal_credentials

def process_withdrawer_exit(state: BeaconState, withdrawer_exit: WithdrawerExit) -> None: if is_withdrawer_exit_valid(state, withdrawer_exit): initiate_validator_exit(state, withdrawer_exit.validator_index)

What's wrong or unclear about it?

1. We probably want the amount of voluntary plus withdrawer exits to be under the same limit MAX_VOLUNTARY_EXITS

. They are both still "voluntary" in the sense that not forced by the protocol for misbehavior.

1. Check for validity validate_withdrawer_exit

seems to be cheap enough to permit much more of them per slot. But the size of BeaconBlockBody.withdrawer_exits

cannot be extended because all its items are supposed to be either discarded as invalid or fulfilled in this block.

A possible way out is to implement a larger amount of validity checks is to validate withdrawer exit requests on the way to BeaconBlockBody.withdrawer_exits

. Here we assume there is a preliminary queue alike the queue for deposit events. The validity check is the same: call to is_withdrawer_exit_valid(...)

.

The Beacon spec doesn't fully cover the question of how BeaconBlockBody.deposits

gets populated, so let's turn to Prysm client implementation for an example.

The BeaconBlockBody.deposits

array is formed in file proposer_deposits.go

in function deposits(...)

(the code).

Let's assume params.BeaconConfig().MaxWithdrawerExitsPerBlock

equals to MAX_WITHDRAWER_EXITS

. And params.BeaconConfig().MaxWithdrawerExitsChecksPerBlock

is max validity checks per block. Then pseudocode of the function to populate BeaconBlockBody.withdrawer_exits

could look like this.

func (vs *Server) withdrawer_deposits(ctx context.Context, beaconState state.BeaconState, currentVote* ethpb.Eth1Data,) ([]*ethpb.Deposit, error) { // ...

```
var pendingExits []*ethpb.WithdrawerExitContainer
for i, exit := range allPendingContainers {

    // check of `exit` request validity as it's descripbed in Beacon spec `is_withdrawer_exit_valid`, where `isExitValid` function does the check
    if isExitValid(exit) {
        pendingExits = append(pendingExits, exit)
    }

    // Don't do more than the max allowed amount of checks
    if i == params.BeaconConfig().MaxWithdrawerExitsChecksPerBlock {
        break
    }

    // Don't try to pack more than the max allowed in a block
    if uint64(len(pendingExits)) == params.BeaconConfig().MaxWithdrawerExitsPerBlock {
        break
    }

}
// ...

}
```

So, regarding the desired implementation on the CL client, there are two questions:

1. What is the max amount of is_withdrawer_exit_valid

checks per slot from a client performance point of view?

1. Is it reasonable to do the validity checks before the requests get to BeaconBlockBody.withdrawer_exits

?

Security considerations

There is an attack surface when the withdrawer exit request queue gets clogged for a long time by spamming with invalid requests.

What might be the attack targets?

- Blackmail attack by a validator(-s) threatening to penalize the owner funds
- Attack on staking pools
- Attack on the entire Ethereum protocol

Blackmail attacks by validators seem to be too expensive even for a group of validators.

What happens to the protocol if honest withdrawer exits get blocked for weeks?

It’s not good but not the end of the world. Validator-initiated exits are still available.

It seems that staking pools with untrusted validators are at the most risk. The specific risks and their conditions seem to be much dependent on the pool implementation.

Questions

1. Should withdrawer exits be added to Shanghai along with the withdrawals?
2. Do you consider the Generalized Message Bus approach worth further development?
3. Are there attacks not outlined in the post?
4. How expensive withdrawer exit validity checks are? Is there a reasonable way to implement them before populating BeaconBlockBody.withdrawer_exits

?

1. Better “withdrawer exit” naming?