Insecura - my consensus for the pyrmont network

TL;DR: pyrmont long range attack howto

: this post is based on a note that might be more up to date with regards to the access instructions, as this is part of a testing regime for Nimbus.

The assumption of weak subjectivity in Ethereum is that anyone wanting to join the network is sufficiently aware of what's going in the world on to be able to tell that some recent point in the history of the chain is the canonical one through social means, for example by asking a friend for a hash, checking checkpoints on web pages or newspapers and so on.

However, hashes are tricky, and so are majority-of-nodes decisions and other tin-foil-hat measures - it's much nicer to just copy-paste a URL and hope it will be fine: say hello to checkpoint sync - pioneered by Teku and now being added to all clients, Nimbus included!

The security assumptions around checkpoint sync are similar toTOFU as seen in SSH: we assume users have access to an oracle they trust the first time they interact with the network, and from there onwards the economic incentives take over: another way to phrase that is that checkpoint sync reduces the security of eth2 for a joining (or rejoining, in the case of prolonged abcense) node to a single point of failure - the URL of the trusted node. Although this doesn't have to be the user experience, it currently by and large is.

In contrast to SSH, verifying a fingerprint (or hash, in this case) is generally not part of the expected checkpoint UX - the videos and tutorials about checkpoint sync basically say "just make it work in 2 minutes".

The long range attack

There are papers, wikis, vitalik-posts and other sources describing the class of attack that we're about to pull off.

It works more or less as follows:

- A set of private keys are compromised somehow - either the validators have exited and the now-useless keys are sold on a secondary market or the keys find their way into the wrong hands, for example after a leak at a large custodial staking operator
- These private keys are used to generate a new history from an arbitrary point in time
- In this history, only the compromised keys are attesting
- The liveness mechanism in eth2 kicks in and exits the "canonical" validators from the state
- The compromised validators now have quorum to finalise the chain and create a new, alternate reality

How checkpoint sync makes things worse

When users download a client, it comes with a set of parameters to bootstrap it: a genesis state as computed from the deposit contract and a set of bootstrap nodes that allow a joining client to discover other peers on the network. There's a common source of well-known bootstrap nodes that are included in client releases, and connections to these are verified by their public key.

From the bootstrap nodes, clients connect to other nodes on the network somewhat randomly, and perform a Status

exchange to see if the peer they just connected to is following a potentially viable chain - if it is, the client starts downloading blocks and verifying them.

Should the client encounter a malicious node, it could be tricked into following a chain that is in the middle of of a long range attack for a while - this chain will not be finalising along with the rest of the network, and other nodes the client connects to will give conflicting histories - above all, if less than 1/3 of the keys at the time of the fork are compromised, the canonical nodes will be serving a much better, finalising history while the other chain has a long period of non-finality as the honest validators are being leaked.

If a single attacker controls >2/3 of all keys that were active at the point of the hostile fork, things get murkier - here, they can create a chain that is equally attractive, and the choice will therefore depend on which chain you first observe as finalising, which in turn depends on which peers you're connected to. There is still an element of chance involved - if the node is not being eclipsed, it will likely choose the canonical history simply because there are more peers serving it.

Checkpoint sync on the other hand teaches users to pass a single URL of REST endpoint they trust to the beacon node. The beacon node downloads a recent state from that URL, then syncs as usual, to catch up with the other nodes.

When this URL is compromised, the attacker can feed the client any state, and the client will "believe" it as long as it passes some basic sanity checks. In particular, it can give them a state that has finalised a different point than where the canonical validators are.

When a node is syncing from a compromised state, it will end up rejecting

the "canoncial" peers in the status exchange and when it asks for blocks, and accept only compromised or dishonest peers, becuase the alternate chain is finalised at a point that cannot be reconciled with the canonical chain.

The insecura network

On the pyrmont network, there are ~120k validators - a malicious duck has taken control of 12000 of them, and is now generating its own little consensus world, called insecura!

To achieve finality (to make the attack look credible), the effective stake of our 12000 validators needs to become the 2/3 majority, meaning that we need to bleed out a large enough part proportion of the other ~108k validators by first leaking their balance, then having them go through the exit procedure - the exact number of validators that must be leaked depends on their balance: their attestation and block production work is weighted by it.

During non-finality, an inactivity leak starts taking effect, slowly increasing the penalty that each inactive validator is subjected to for each epoch, significantly speeding up their eventual exit. Math is involved, but effectively, it takes a few thousand epochs to reduce their balance to the requisite 16ETH.

Once the balance of the validators is sufficiently, it takes another 25000 epochs to have the 100k validators exit, due to the exit churn limit: at most 4 validators are exited every epoch - this doesn't matter that much for the attack - all we need is 2/3 of the staking balance (not validator counts).

There are two ways to carry out a long range attack:

- use only exited keys - this is the safer option, but it obviously takes a lot of time
- use keys that have not exited the "honest" fork, instead "double-voting" on the new chain

When using the latter option, we will be creating a history parallel to the "canonical" chain - in particular, this means we'll be creating attestations on both forks which normally is a slashable offence. This is risky business - if we stop attesting on the canoncial chain however, the risk is contained as clients typically will not look for slashable offences in past blocks. The risk is further mitigated by the difficulty of detecting that the attestations are indeed duplicate: followers of the canonical chain will not have the shuffling of the forked chain because they will not have accepted the blocks into their history. Likewise for blocks: it's likely that duplicate blocks get dropped by honest clients before they reach any slashers: without access to the parent, they are difficult to validate.

On altair, and other stumbling blocks…

When recreating a state using a subset of the validators, phase0 rules dictate that all validators are affected by an inactivity leak. At the same time, there's a limit to how many validators can exit per epoch. This leads to an interesting effect:

- Even when controlling a significant amount of validators - say 15%, the chain will die because these validators will lose too much balance before the other validators have been ejected and the chain can finalize
- The chain finalizes even though a supermajority of validators are not attesting
- when the balance of each validator has been reduced to 0, you don't need any balance at all to finalize the chain.

Altair makes it easier to carry out this attack: validators that are attesting don't leak much more slowly.

Step by step

To follow along, you'll need a pyrmont-synced copy of nimbus - an excellent client to make these kinds of experiments.

Armed with this knowledge and a premonition of the imminent demise of pyrmont, which at the time of writing is at epoch 94457, we create a fork at epoch 90000 - we'll use the last block in the preceding epoch as pivot, advanced to the first slot of the new epoch:

Compile everything that's needed

git clone https://github.com/status-im/nimbus-eth2.git

cd nimbus-eth2 git checkout unstable # or maybe wss-sim at time of writing make update -j4

source ./env.sh cd ncli nim c -d:release ncli_db nim c -d:release ../research/wss_sim

Take a snapshot of a synced pyrmont directory, validators and all

cp -ar pyrmont_0 insecura_0 cd insecura_0

Generate a starting state for wss_sim

../ncli_db --network:pyrmont --db:db rewindState 0x6134fbda3713c25f8c69450350e648488b2cbc9564f20e1c1ef82d838747fccf 1920000

Run wss_sim with the validators residing in the same folder and the state that was generated by ncli_db - note how it still uses the pyrmont genesis

../research/wss_sim --network:pyrmont --validatorsDir:validators/ --secretsDir:secrets/ --startState:state-1920000-6134fbda-5d211b58.ssz

Import the states and blocks - there's lots of them so we have to use find

../ncli_db --network:pyrmont --db:db putState state-*find -name "block-*" -print0 | xargs -0 ../ncli_db --network:pyrmont --db:db putBlock

Look for the *last* block and set it as head

../ncli_db --network:pyrmont --db:db putBlock block-123.ssz --set-head

You can now launch a nimbus beacon node with the new database and serve REST states to anyone that wants them! Muahahaha! Quack Quack! This node is pretty cool: not only is it hosting 12000 validators - it also takes no more than a gig of memory and runs on a single CPU. If ever you wondered how Vitalik possibly could be running his master node, this is it.

nimbus_beacon_node --data-dir:. --network:pyrmont --rest --sync-horizon:100000

Where to go from here

The way this particular issue arises can be detected in a number of ways - there are multiple red flags along the way:

- A large chunk of validators exit the chain via the inactivity leak
- This is a strong telling sign that something is going on - in fact, this is perhaps the best way to tell that the client is not on a canoncial chain: it's the only way that these validators can lose their voting power without

their own input / signature

- This is a strong telling sign that something is going on - in fact, this is perhaps the best way to tell that the client is not on a canoncial chain: it's the only way that these validators can lose their voting power without

their own input / signature

- A majority of nodes you encounter are following another chain
- this alone is not enough to discard the majority chain - it may be that you have bad luck or are unable to connect to honest nodes

- this alone is not enough to discard the majority chain - it may be that you have bad luck or are unable to connect to honest nodes

- In the two histories, there are overlapping votes
- Since the "alternate" history generally is not checked by honest clients, they don't detect this condition through "normal" means

- Since the "alternate" history generally is not checked by honest clients, they don't detect this condition through "normal" means

@djrtwo has posted some further information about this problem:WS sync in practice - HackMD

FAQ

Why 12000 validators?

For the chain to run, validators need to be producing blocks and attestations - however, without blocks, attestations are not included in the chain - even if validators are online and performing their attestion work, there needs to be a sufficent amout of block producers or blocks become infrequent enough that the produced attestations no longer fit - the end effect is the same as if the validators were offline.

The other aspect is that validators lose balance when the chain is not finalising - much more so in phase0 - therefore, even if the validators you have are not bleeding as fast as the others, they still risk ending up below 16 ETH before

the chain finalises, causing them to be ejected even if they're the only ones left doing work.

That said, it's probably possible to do with fewer, specially post-altair when non-finality penalties are lower.

How long did it take to generate the history?

About 2 days on a single thread - the simulator is creating attestations for all validators it has keys for, then packing these into blocks - the signature part could easily be parallelised.

Why not start from a fully constructed state with only brand new validators?

Using parts an existing history lends some legitimacy to the chain - in particular, it looks plausible from a “full sync” perspective - there’s nothing going on in this history which violates the protocol, consensus or anything else.

What happens with deposits?

When the chain finalizes, deposits will be processed as “normal”.

What are some peer ENR:s I can use as bootnodes?

Eventually, you will find these peers via discovery, but if you want to get connected more quickly, these can be used as boot nodes:

What’s the current state of the network

Check it via the REST API

ENR:

enr:-LK4QGQl-
6vfK7JdE8zPVzk9rMrXI1myBuy9xFZfA4JgEJm3ZScaoyyfPy3t6X57tY2G7gLK9zMzSkuFI1Hf0kRv7r4Bh2F0dG5ldHOIAAAAAAAAAACEZXRoMpB0ZesLAQAgCf_***gmlkgnY0gmlwhEEVxC2Jc2VjcDI1N***
enr:-LK4QKbOD5MlwM**BE7uVLwpRlwhlsFxiiOyReRhUBc7jgzblf8EXG5MccDxuRChfRqthkSo4wwC_ICECy2u8QVjx1esBh2F0dG5ldHOIAAAAAAAAAACEZXRoMpB0ZesLAQAgCf**_gmlkgnY0gmlw
szjqex-IuG7NeNcIhp6jvc8j-hsFnNi14a9FTYGToN0Y3CCIyqDdWRwgiMq enr:-LK4QNymCG1DTf2rrVqzz9yTwU3-
lE8_qrufLtUmlGU7nlsib4QNEpr_Csm_hfAjnjYL7uDcjTRkLLti9lwZH_UxnaQBh2F0dG5ldHOIAAAAAAAAAACEZXRoMpB0ZesLAQAgCf____gmlkgnY0gmlwhEEVxC2Jc2VjcDI1NmsxoQMZ5KZLn-
wqfc1FBWmeIn2lqsMmeUPI7nCerJQuBrOzLIN0Y3CCIymDdWRwgiMp

libp2p:

16Uiu2HAmKXzMg7pJySEGyheb7MFq5TodkdNQCmYydFAaJJHMB9aa 16Uiu2HAmNkSuYNcstMquNnetkytweScik3XWLB5NWKEjYktUTiS1
16Uiu2HAmEQ4m9PKLHDQu3bqSNJuTWBTiFkQA3drzXm33oYiCbeCT

Client command lines

- Nimbus

build/nimbus_beacon_node --network:pyrmont --data-dir:$DATADIR trustedNodeSync --trusted-node-url:http://insecura.nimbus.team --backfill:false

build/nimbus_beacon_node --network:pyrmont --data-dir:$DATADIR

- Teku

bin/teku --initial-state=http://insecura.nimbus.team/eth/v2/debug/beacon/states/finalized --network=pyrmont --data-path=/data/teku_ins --p2p-discovery-bootnodes=enr:-
LK4QNymCG1DTf2rrVqzz9yTwU3-
lE8_qrufLtUmlGU7nlsib4QNEpr_Csm_hfAjnjYL7uDcjTRkLLti9lwZH_UxnaQBh2F0dG5ldHOIAAAAAAAAAACEZXRoMpB0ZesLAQAgCf____gmlkgnY0gmlwhEEVxC2Jc2VjcDI1NmsxoQMZ5KZLn-
wqfc1FBWmeIn2lqsMmeUPI7nCerJQuBrOzLIN0Y3CCIymDdWRwgiMp --p2p-static-peers=/ip4/65.21.196.45/tcp/9001/p2p/16Uiu2HAmEQ4m9PKLHDQu3bqSNJuTWBTiFkQA3drzXm33oYiCbeCT

- Lighthouse

curl -o state.ssz -H 'Accept: application/octet-stream' http://insecura.nimbus.team/eth/v2/debug/beacon/states/finalized curl -o block.ssz -H 'Accept: application/octet-stream'
http://insecura.nimbus.team/eth/v2/beacon/blocks/finalized ./lighthouse beacon_node --datadir=/data/lh-ins --network=pyrmont --checkpoint-block=block.ssz --checkpoint-state=state.ssz --boot-
nodes=/ip4/65.21.196.45/tcp/9001/p2p/16Uiu2HAmEQ4m9PKLHDQu3bqSNJuTWBTiFkQA3drzXm33oYiCbeCT

- Lodestar

./lodestar beacon --network pyrmont --eth1.enabled false --rootDir /data/insecura --weakSubjectivitySyncLatest --weakSubjectivityServerUrl insecura.nimbus.team --network.discv5.bootEnrs enr:-
LK4QNymCG1DTf2rrVqzz9yTwU3-
lE8_qrufLtUmlGU7nlsib4QNEpr_Csm_hfAjnjYL7uDcjTRkLLti9lwZH_UxnaQBh2F0dG5ldHOIAAAAAAAAAACEZXRoMpB0ZesLAQAgCf____gmlkgnY0gmlwhEEVxC2Jc2VjcDI1NmsxoQMZ5KZLn-
wqfc1FBWmeIn2lqsMmeUPI7nCerJQuBrOzLIN0Y3CCIymDdWRwgiMp

- Prysm

TBD