BeaconFuzz

A Journey into Ethereum 2.0 Blockchain Fuzzing and Vulnerability Discovery

FUZZING LABS

Whoami

Patrick Ventuzelo / @Pat_Ventuzelo

- Twitter / LinkedIn / Github / Blog / Youtube

Founder of **FuzzingLabs** | Senior **Security Researcher**
⇒ Training/Consulting

Previously:

- QuoScient GmbH
- P1 Security
- French DoD
- Airbus Defense & Space

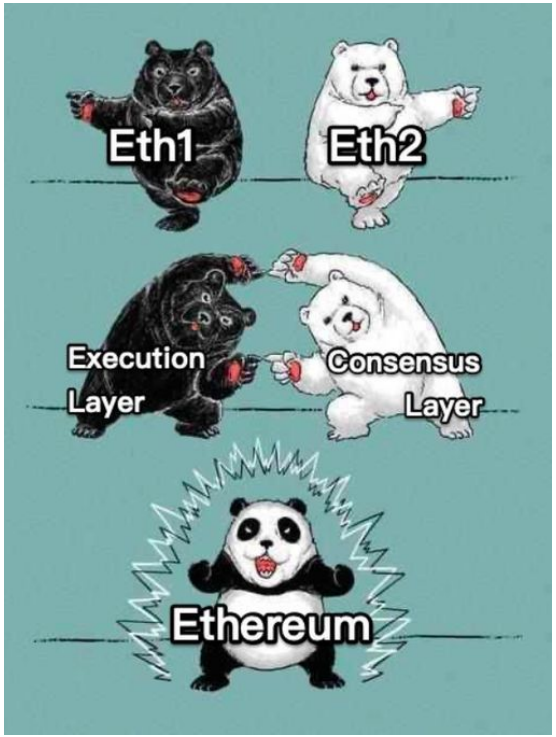
⇒ Fuzzing, Vulnerability research
⇒ Rust, Golang, WebAssembly, Browsers
⇒ Blockchain Security, Smart contracts

Ethereum 2.0 Blockchain

What's Ethereum & Ethereum 2.0 ?

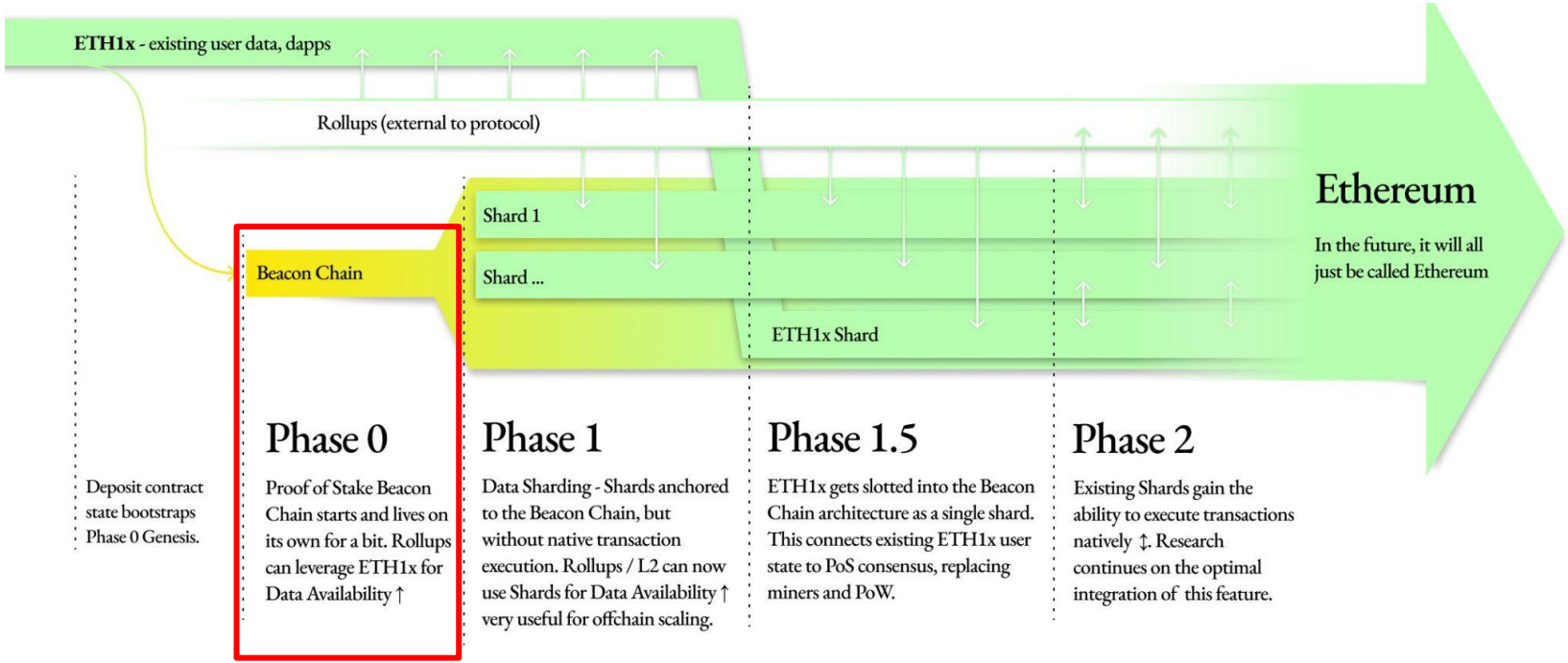
- What's **Ethereum**?
  - The first blockchain running Smart Contracts
    - Created in 2015
  - Networking: **Peer-to-peer (P2P)**
    - Decentralized platform
  - Smart contracts
    - Mostly written in Solidity
    - Compiled in EVM bytecode
  - Consensus: **Proof Of Work (PoW)**
    - Miners compete to append blocks and mint new currency
    - Energy consumption is huge

- **Ethereum 2.0** (new naming: **Ethereum Consensus**)
  - Upgrade of Ethereum
  - **New consensus: Proof of Stake (PoS)**
    - Selecting validators in proportion to their quantity of tokens holdings
    - Validators are replacing miners
  - Not a new blockchain but an **evolution**



FUZZING LABS

Ethereum 2.0 Roadmap

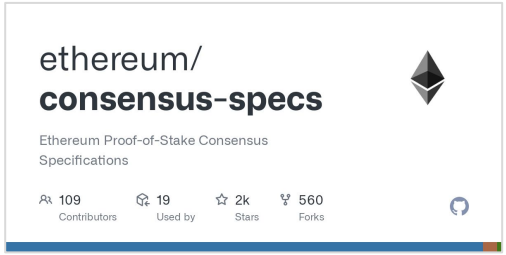


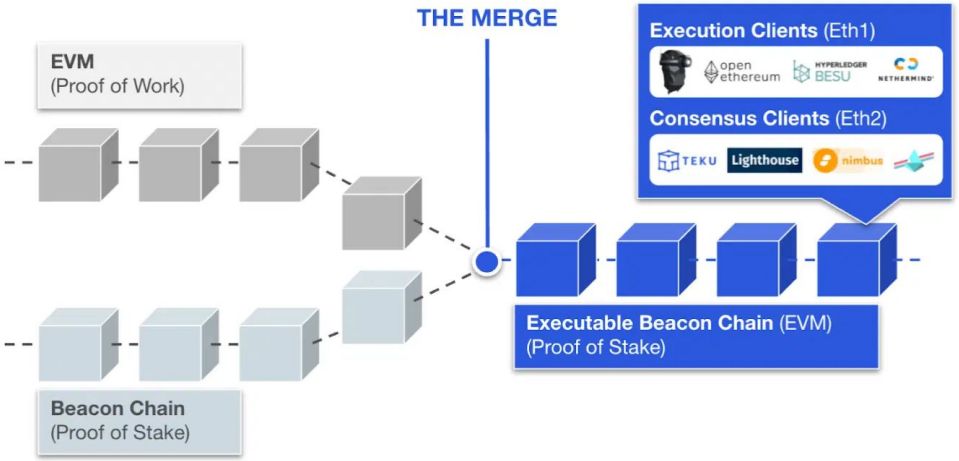
FUZZING LABS

What's Ethereum 2.0 Beacon chain?

- **Specification**
  - Ethereum PoS Consensus
    - Written in Python
    - With documentation + unit tests
  - Followed by all ETH2.0 clients
  - Divided into Phase & features
    - Researched and developed in parallel
  - Released over time
    - **Dec 2021 - Phase 0**
    - Q4 2021 - Altair
    - Q2 2022 - Bellatrix (aka The Merge)
    - 2022/2023 -Sharding

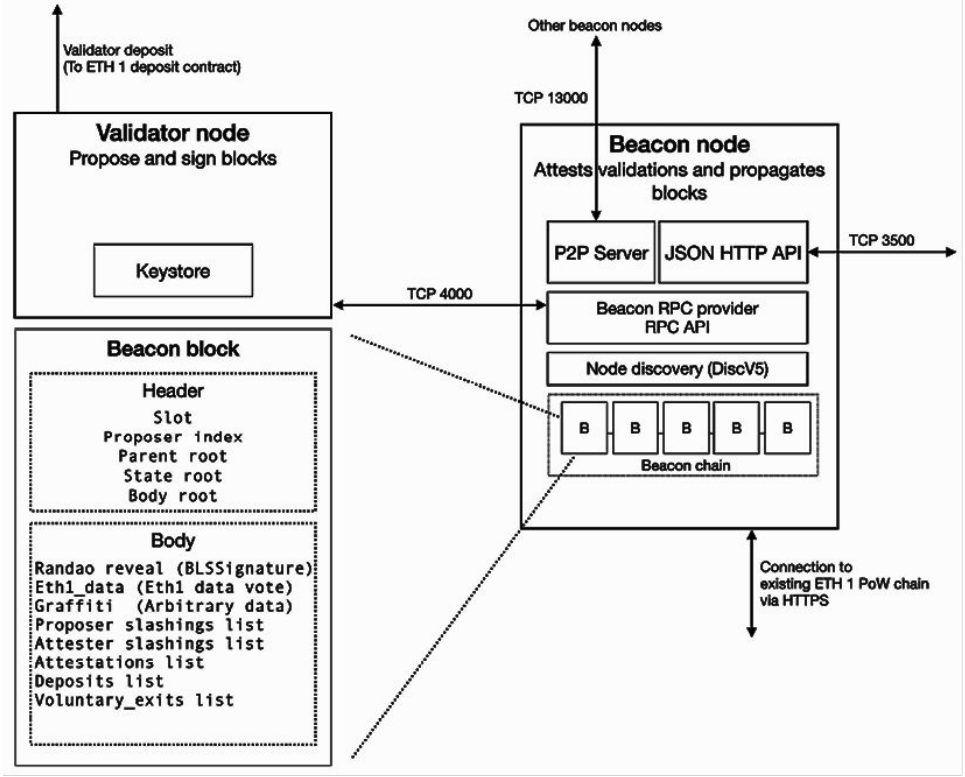
- **Phase 0 - Beacon chain**
  - The Core of Ethereum proof-of-stake
  - Stores and manages the registry of validators
  - Beacon Chain specification - link





ETH2.0 Client - Architecture & Attack Surface

- 2 Separate binaries
  - Validator client
  - Beacon node
- **Networking stack** (libp2p & devp2p)
  - ENR: Ethereum Node Records
  - Discv5: Discovery Protocol v5
  - Gossipsub, etc.
- **State transition logic**
  - Simple Serialize (SSZ) objects
    - Decoding/encoding
  - Beacon Block & other Datatype processing
- Which kind of bugs are interesting?
  - Crashes/Panics
  - Memory corruption, Denial Of Service (DoS)
  - Consensus/Logic bugs



ETH2.0 Client - Consensus clients overview

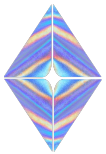
- Lighthouse
  - Developed by Sigma Prime
  - Written in **Rust**
- Prysm
  - Developed by Prysmatic Labs
  - Written in **Go**
- Nimbus
  - Developed by Status
  - Written in **Nim**
- Teku
  - Developed by ConsenSys
  - Written in **Java**
- Lodestar
  - Developed by ChainSafe
  - Written in **TypeScript/JavaScript**

Lighthouse

Prysm

Nimbus

Teku



Lodestar

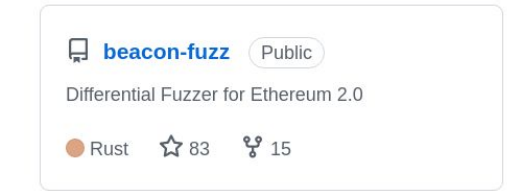
Beacon Fuzz Roadmap

Beacon Fuzz History

- May 2019 - **Guido's eth2 fuzzing**
  - The Ethereum Foundation engaged **Guido Vranken** to build differential fuzzing across existing Ethereum 2.0 clients.
  - eth2.0-fuzzing (written in C++) leverages **libFuzzer** to provide the same fuzzing input on all targets.
    - focussed on fuzzing ZRNT and Pyspec, the **Go and Python executable** Ethereum 2.0 specification.

- September 2019 - **Resumption** by **Sigma Prime**
  - Sigma Prime received a grant from the Ethereum foundation to continue the project.
  - **Maintainability** of the differential fuzzing platform.
    - Upgrading the fuzzing targets to match the **latest version** of the Ethereum 2.0 specification.
    - Support more clients and create a set of valid inputs (corpora) to be used by the differential fuzzer.

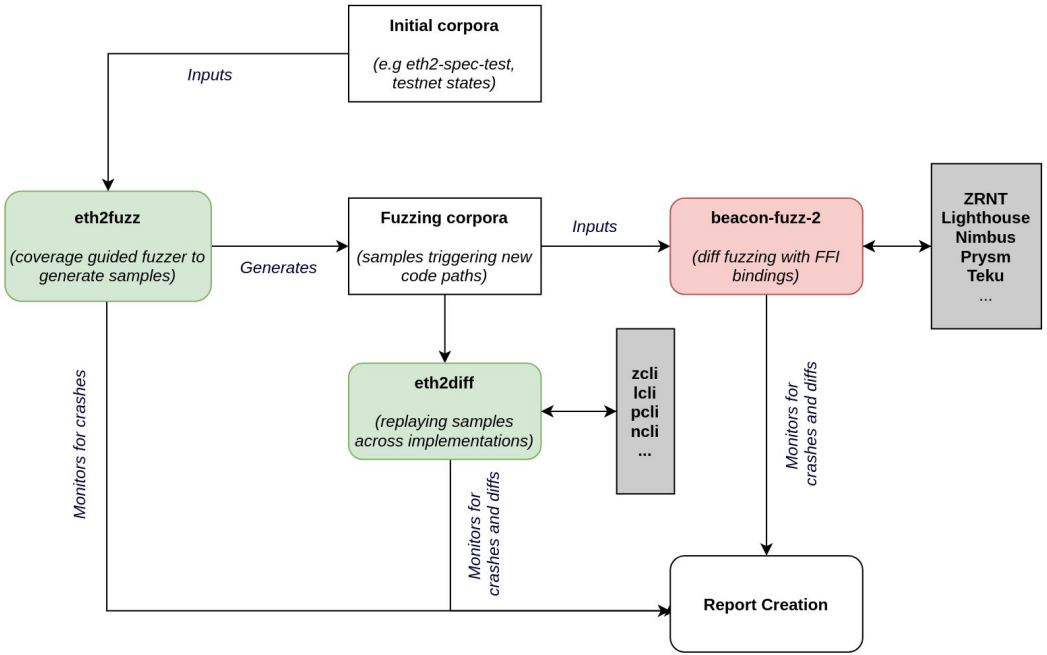
- March/April 2020 - Start of **Beacon Fuzz** project
  - Fuzzinglabs & Sigma Prime start working together.
  - Exploring other options to achieve the same goals.
    - **Improve the coverage & find new bugs.**



⇒ This talk represents in part the work done periodically over the past 1-2 years

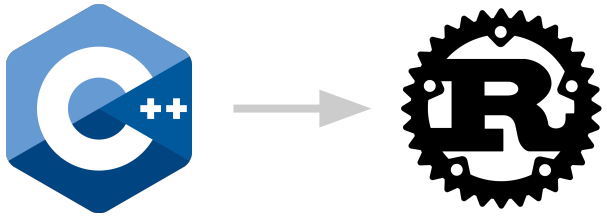
Beacon Fuzz Roadmap

- Understand the **context**
  - Learn more about Ethereum 2.0 PoS
  - Compiling & testing all projects
- Create a testing/fuzzing **corpora**
  - using the eth2 specification
  - using all clients unit tests
- Create multiple fuzzing tools
  - **Coverage-guided fuzzer** (eth2fuzz)
  - Simple differential fuzzer (eth2diff)
  - Structural fuzzing implementation
  - **Differential fuzzer** (beacon-fuzz-2)
- Integrate **new clients**/targets
  - Start with the more up-to-date clients
  - Simplify the compilation with docker



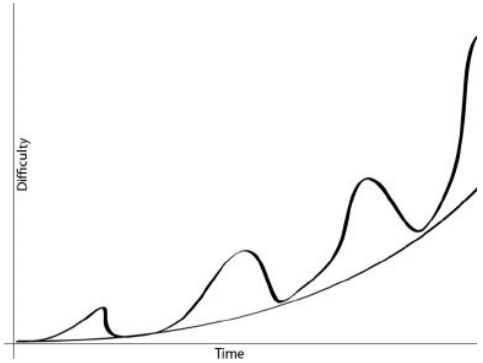
Beacon Fuzz Roadmap - Design & Choices

- Why rewrite everything in Rust?
  - Ease of development & maintainability
  - Better tooling & ecosystem
  - Foreign Function Interfaces (FFI) bindings
  - Structural fuzzing is easier with **Arbitrary trait**
  - (also I REALLY don't like C/C++)



- Why create multiple fuzzing tools?
  - They are not all targeting the same code
  - Some of them will be **faster to keep up-to-date**
  - They do not require the same computer power for compilation & fuzzing
  - They do not always follow the same **specification** version & clients branches

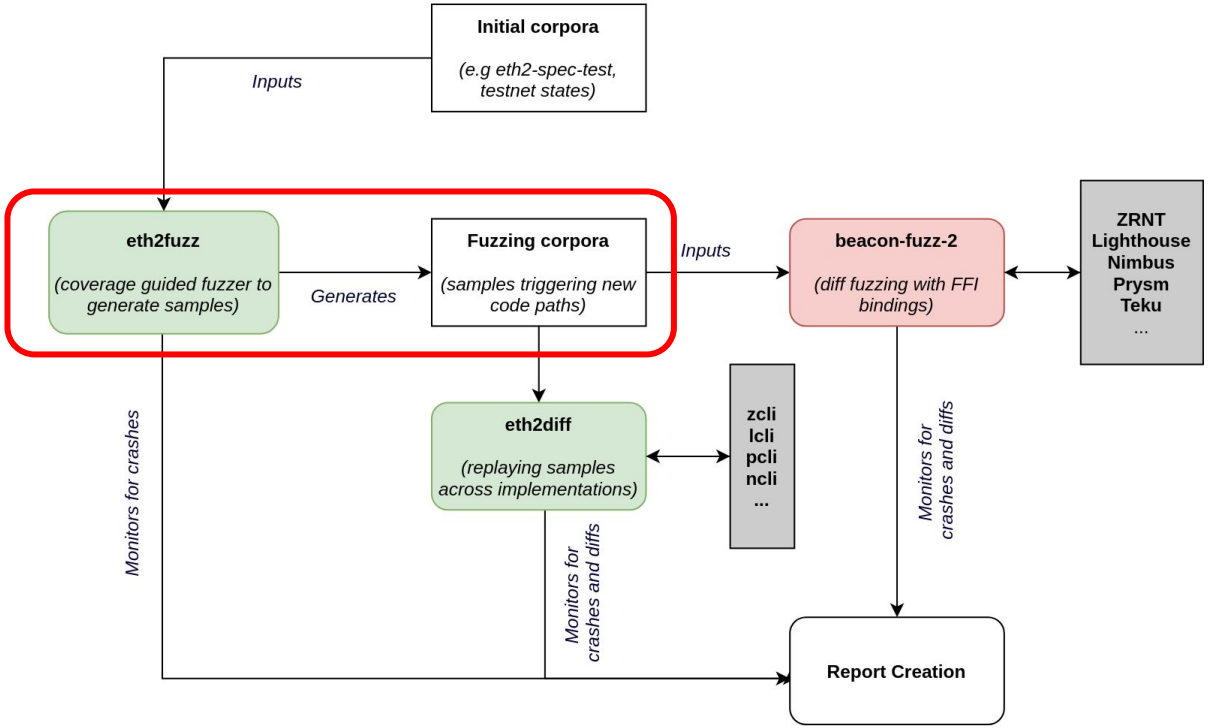
- Why not directly use the most evolved/efficient fuzzing techniques?
  - Simple fuzzer will catch **faster** potentially blocking **low hanging-fruits**
  - It's better to increase the development complexity over time
    - to prevent being overcharged by difficulty
    - to adapt to new information or clients updates



FUZZING LABS

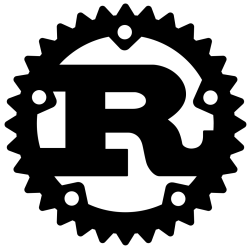
eth2fuzz - Coverage-guided fuzzing

eth2fuzz - Coverage-guided fuzzing on all clients



eth2fuzz - **Lighthouse** coverage-guided fuzzing

- Lighthouse - Rust Ethereum 2.0 Client
  - Maintained by Sigma Prime
  - Written in **Rust**



- Rust fuzzers
  - hfuzz-rs: Fuzz your Rust code with Google-developed Honggfuzz
  - cargofuzz: A cargo subcommand for fuzzing with libFuzzer
  - afl-rs: Fuzzing Rust code with AFLplusplus

- Complexity: **Low/Medium**
  - Generation of an SSZ binary
  - Decoding of the SSZ into a valid structure
  - Loading randomly one valid Beacon State
  - Processing of the state transition

```rust
/// Run `process_block_header`
pub fn process_header(mut beaconstate: BeaconState<MainnetEthSpec>,
    block: BeaconBlock<MainnetEthSpec>)
        -> Result<(), BlockProcessingError> {
    let spec = MainnetEthSpec::default_spec();

    process_block_header(&mut beaconstate, &block, &spec)?;

    Ok(())
}
```

FUZZING LABS

eth2fuzz - **Lighthouse** results

- **3 Bugs found**
  - Memory allocation failure in SSZ decoding due to OOB of variable-length types - link
    - Vec::with_capacity called with an unchecked size argument
  - Panic when decoding non-utf8 string as an ENR - link
  - Panic due to multiplication overflow when getting the Beacon proposer index of BeaconState - link
    - When Rust code is compiled in debug mode, overflows are checked and triggered panics.
    - Beacon State was considered as a trusted container
      - leads to clarification and update regarding the overflow assumptions of the eth2 specification - link

```
let effective_balance = self.validators[candidate_index].effective_balance;
if effective_balance * MAX_RANDOM_BYTE >= spec.max_effective_balance * u64::from(random_byte)
{
    return Ok(candidate_index);
}
```

```
let mut values = Vec::with_capacity(num_items);
// Only initialize the vec with a capacity if a r
//
// We assume that if a max length is provided the
// allocation of this size.
let mut values = if max_len.is_some() {
    Vec::with_capacity(num_items)
} else {
    vec![]
};
```

- Limitation: **None**
- Possible improvement
  - Structural fuzzing using **Arbitrary trait**
  - Add more fuzzing harnesses
    - Increase code coverage

FUZZING LABS

eth2fuzz - **Prysm** dumb fuzzing

- Prysm - Go implementation of Ethereum proof of stake
  - Maintained by Prysmatic Labs
  - Written in **Go**

- Interesting utility/testing tools
  - pcli state-transition: Subcommand to run manual state transitions

```
bazel run //tools/pcli:pcli -- state-transition --block-path /path/to/block.ssz --pre-state-path /path/to/state.ssz
```

- Complexity: **None**
  - Basic shell script
  - Replay inputs generated during lighthouse fuzzing

- Bugs found: **None**

eth2fuzz - **Prysm** coverage-guided fuzzing

- Go fuzzers
  - go-fuzz: Randomized testing for Go
  - libfuzzer: Generate an archive file
    - that can be used with libFuzzer

- Complexity: **Medium**
  - We can't use classical usage of go-fuzz for Prysm
    - Herumi's cgo-based BLS implementation
    - go-fuzz doesn't support cgo - link
  - They were using Bazel for building
    - Bazel is painful if you're not trained
  - We asked for native go build integration
    - i.e. make Prysm "go gettable"
    - "biggest feature of the year"
      - For Prysm external contributor

```go
func Prysm_block_header(b []byte) int {
    params.UseMainnetConfig()
    data := &ethpb.BeaconBlock{}
    if err := data.UnmarshalSSZ(b); err != nil {
        return 0
    }
    // get a valid beaconstate
    s, err := stateTrie.InitializeFromProto(GlobalBeaconstate)
    if err != nil {
        // should never happen
        panic("stateTrie InitializeFromProto")
    }
    // process the container
    post, err := blocks.ProcessBlockHeaderNoVerify(s, data)
    if err != nil {
        return 0
    }
    if post == nil {
        return 0
    }
    return 1
}
```

eth2fuzz - **Prysm** results

- **3 Bugs found**
  - Slice bounds out of range when parsing SSZ - link
  - Nil pointer dereference when processing ProposerSlashing - link
  - Slice bounds out of range when parsing SSZ #2 - link

```
./prysm_FuzzProposerSlashing2.libfuzzer panic_nil_deref_prysm_proposer.ssz
INFO: Seed: 1287398888
./prysm_FuzzProposerSlashing2.libfuzzer: Running 1 inputs 1 time(s) each.
Running: panic_nil_deref_prysm_proposer.ssz

panic: runtime error: invalid memory address or nil pointer dereference
[signal SIGSEGV: segmentation violation code=0x1 addr=0x38 pc=0x11d0360]
```

- Limitation: **None**
- Possible improvement
  - Structural fuzzing using gofuzz

```go
package main


import (

    "fmt"

    "github.com/prysmaticlabs/prysm/shared/params"
    "github.com/prysmaticlabs/prysm/beacon-chain/p2p/encoder"
    testpb "github.com/prysmaticlabs/prysm/proto/testing"
)


func DecodeTestSimpleMessageCrash() {
    data := []byte("\x01\x00\x8f")
    params.UseMainnetConfig()
    input := &testpb.TestSimpleMessage{}
    e := encoder.SszNetworkEncoder{}
    if err := e.DecodeGossip(data, input); err != nil {
        _ = err
        return
    }
    return
}


func main() {
    fmt.Println("prysm: Crash reproducer")
    // change the following function to trigger different bugs
    DecodeTestSimpleMessageCrash()
}
```

FUZZING LABS

eth2fuzz - **Nimbus** dumb fuzzing

- Nimbus - Nim implementation of the Ethereum 2.0 blockchain
  - Maintained by Status
  - Written in **Nim**
- Interesting utility/testing tools
  - ncli_pretty: Pretty-print **SSZ object** as JSON
  - ncli_transition: Perform **state transition**
    - given a pre-state and a block to apply
  - ncli_hash_tree_root: Print tree root of an **SSZ object**
- Complexity: **None**
  - Basic shell script
  - **Replay** inputs generated during lighthouse fuzzing
- **5 Bugs found**
  - Segmentation fault during State transition - link
  - AssertionError during State transition - link
  - IndexError during Attestation SSZ parsing - link
  - IndexError during Beaconstate SSZ parsing - link
  - IndexError during Beaconstate SSZ parsing #2 - link

```
cli do(pre: string, blck: string, post: string, verifyStateRoot = true):
  let
    stateY = (ref HashedBeaconState)(
      data: SSZ.loadFile(pre, BeaconState),
    )
    blckX = SSZ.loadFile(blck, SignedBeaconBlock)
    flags = if not verifyStateRoot: {skipStateRootValidation} else: {}

  stateY.root = hash_tree_root(stateY.data)
```

```
$ ./ncli_hash_tree_root --kind=attestation --file= IndexError_attestation_empty_containe
Traceback (most recent call last, using override)
XXX/nim-beacon-chain/vendor/nim-confutils/confutils.nim(981) confutils
XXX/nim-beacon-chain/ncli/ncli_hash_tree_root2.nim(14) CLI
XXX/nim-beacon-chain/beacon_chain/ssz.nim(583) hash_tree_root
XXX/nim-beacon-chain/beacon_chain/ssz.nim(444) hashTreeRootImpl
XXX/nim-beacon-chain/beacon_chain/ssz.nim(570) hash_tree_root
XXX/nim-beacon-chain/beacon_chain/ssz.nim(466) bitlistHashTreeRoot
XXX/nim-beacon-chain/vendor/nimbus-build-system/vendor/Nim/lib/system.nim(2515) X5BX5D
XXX/nim-beacon-chain/vendor/nimbus-build-system/vendor/Nim/lib/system/chcks.nim(23) rais
XXX/nim-beacon-chain/vendor/nimbus-build-system/vendor/Nim/lib/system/fatal.nim(51) sysF
XXX/nim-beacon-chain/vendor/nimbus-build-system/vendor/Nim/lib/system/excpt.nim(407) rep
XXX/nim-beacon-chain/vendor/nimbus-build-system/vendor/Nim/lib/system/excpt.nim(358) rep
Error: unhandled exception: index out of bounds, the container is empty [IndexError]
```

FUZZING LABS

eth2fuzz - **Nimbus** coverage-guided fuzzing

- Nim Programming Language
  - Syntax similar to Python
  - Compiled language, with strong static typing.
  - Nim compilation process
    - Nim code converted to C
    - C compilation to binary

- Nim fuzzers
  - afl/afl++: template for afl/afl++ abstraction
  - Libfuzzer: template for libfuzzer abstraction
  - Honggfuzz: template for honggfuzz abstraction

```
proc fuzz_nimbus_attestation*(state: var BeaconState, payload: openarray[byte]): bool =
    try:
        var cache = StateCache()
        let attestation = SSZ.decode(payload, Attestation)
        discard process_attestation(state, attestation, {}, cache)
    except SSZError: #CatchableError:
        discard
    true
```

```
# afl-clang
nim c -d:afl -d:noSignalHandler --cc=clang --clang.exe=afl-clang --clang.linkerexe=afl-clang ftestcase.nim
```

- Complexity: **Medium**
  - New language to explore
  - All fuzzers abstraction was not developed at the time
    - But part of Nimbus team are also Nim language core developers

eth2fuzz - **Nimbus** results

- **2 Bugs found**
  - Unhandled exception IndexError when parsing ProposerSlashing - link
  - IndexError during AttesterSlashing processing - link
- Limitation: **None**
- Possible improvement
  - Add more fuzzing harnesses to improve coverage

```
$ ./ncli_pretty --beacon=38542f2a6666ae61361a7d8249eb0a55.ssz --contai
Traceback (most recent call last, using override)
/nim-beacon-chain/vendor/nim-confutils/confutils.nim(981) confutils
/nim-beacon-chain/ncli/ncli_pretty.nim(24) CLI
/nim-beacon-chain/vendor/nimbus-build-system/vendor/Nim/lib/system/exc
/nim-beacon-chain/vendor/nimbus-build-system/vendor/Nim/lib/system/exc
Error: unhandled exception: index 6368 not in 0 .. 255 [IndexError]
```

```
import
  confutils, os, strutils, chronicles, json_serializat
  ../beacon_chain/spec/crypto,
  ../beacon_chain/spec/datatypes,
  ../beacon_chain/spec/digest,
  ../beacon_chain/spec/validator,
  ../beacon_chain/spec/beaconstate,
  ../beacon_chain/spec/state_transition_block,
  ../beacon_chain/ssz,
  ../beacon_chain/extras,
  ../beacon_chain/state_transition,
  ../beacon_chain/eth2_discovery

cli do(beacon: string, container: string):
  try :
    var b = SSZ.loadFile(beacon, BeaconState)
    var c = SSZ.loadFile(container, AttesterSlashing)
    var cache = get_empty_per_epoch_cache()
    discard process_attester_slashing(b, c, {}, cache)
  except SSZError:
    quit 1
  quit 0
```

FUZZING LABS

eth2fuzz - **Teku** dumb fuzzing

- Teku - Java Implementation of the Ethereum 2.0 Beacon Chain
  - Maintained by ConsenSys
  - Written in **Java**
- Interesting utility/testing subcommands
  - teku transition: Manually run state transitions
- Complexity: **None**
  - Basic shell script
  - Replay inputs generated during lighthouse fuzzing
- **5 Bugs found** during Block SSZ parsing
  - DoS/infinite processing - link
  - IllegalArgumentException: List out of bounds - link
  - IllegalArgumentException: Invalid negative length - link
  - IndexOutOfBoundsException: index (-1) must not be negative - link
  - java.lang.IndexOutOfBoundsException: index (0) must be less than size (0) - link


```
# install
./gradlew distTar installDist

# go to build folder
cd build/install/

# Run teku
bin/teku transition blocks  \
    --pre=state.ssz         \
    --network=mainnet       \
    block.ssz
```

```
java.lang.IllegalArgumentException: List out of bounds
    at tech.pegasys.artemis.ssz.SSZTypes.SSZArrayCollect
```

```
java.lang.IndexOutOfBoundsException
```

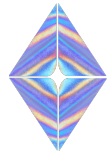
eth2fuzz - **Teku** coverage-guided fuzzing

- Java fuzzers
  - JQF+AFL: Fuzzing a Java program using JQF & AFL
  - Jazzer: Coverage-guided, in-process fuzzing for the JVM
    - (Not available at the time)
- Complexity: **Medium**
  - Java ecosystem to deal with
  - A lot of different exceptions to handle
- **1 Bug found**
  - Illegal Index Array Access in Attester Slashing Processing - link
- Limitation
  - JQF+AFL was **really SLOW** (< 10 exec/s per thread)
  - Complex set up forcing to run the fuzzer inside docker
- Possible improvement
  - Use **Jazzer** as the new fuzzing framework

```
@Fuzz
  public void teku_block_header(InputStream input) {
    Constants.setConstants("mainnet");
    SimpleOffsetSerializer.setConstants();
    try {
      if (TekuFuzz.GlobalBeaconState == null) {
        get_beaconstate();
      }

      byte[] bytes = input.readAllBytes();
      // SSZ deserialization
      BeaconBlock structuredInput =
       SimpleOffsetSerializer.deserialize(Bytes.wrap(bytes)

      // processing container
      TekuFuzz.GlobalBeaconState.updated(
        state -> {
          BlockProcessorUtil.process_block_header(
              state, structuredInput);
        });

    } catch (IOException e) {
    } catch (InvalidSSZTypeException e){
    } catch (EndOfSSZException e){
    } catch (IllegalStateException e){
    } catch (IllegalArgumentException e){
    } catch (BlockProcessingException e){
    }
  }
```

FUZZING LABS

eth2fuzz - **Lodestar** coverage-guided fuzzing



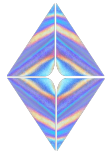
- Lodestar - Ethereum 2.0: TypeScript Implementation of the Beacon Chain
  - Maintained by ChainSafe Systems
  - Written in **TypeScript**
    - Compiled into JavaScript
  - website, github

- Fuzzer
  - jsfuzz: Coverage guided fuzz testing for Javascript

- Complexity: **Low**
  - Direct fuzzing of Lodestar npm package
  - Lodestar APIs are pretty simple
    - And **TypeScript** typing help a lot
  - Jsfuzz harnesses are not complicated as well

```
function fuzz_lodestar_block(buf) {
    var mainnet_1 = require("@chainsafe/lodestar-types/lib/
        ssz/presets/mainnet");
    try {
        mainnet_1.types.BeaconBlock.deserialize(buf);
    } catch (e) {
        is_lodestar_valid_exception(e);
    }
}

function fuzz(data) {
    fuzz_lodestar_attestation(data);
}

module.exports = {
    fuzz
};
```

eth2fuzz - **Lodestar** results



- **7 Bugs found**
  - TypeError in SSZ library during BeaconBlock deserialize - link
  - RangeError in SSZ library when parsing empty BeaconBlock - link
  - "TypeError: public key must be a Buffer" when parsing ENR string - link
  - Memory exhaustion/OOM when parsing invalid ENR string - link
  - "AssertionError" inside bcrypto library when parsing invalid ENR string. - link
  - "Assertion `val->IsArrayBufferView()' failed" when parsing invalid ENR string - link
  - "TypeError: Cannot read property 'toString' of undefined" when parsing ENR string - link

```
var discv5 = require("@chainsafe/discv5");

buf = "enr:-IS4QJ2d11eu6dC7E7LoXeLMgMP3kom1u3SE8esFSWvaHoo0
dP1jg8O3-nx9ht-EO3CmG7L6OkHcMmoIh00IYWB92QABgmlkgnY0gmlwhH8
AAAGJc2d11eu6dCsxoQIB_c-jQMOXsbjWkbN-kj99H57gfId5pfb4wa1qxw
V4CIN1ZHCCIyk".toString()

discv5.ENR.decodeTxt(buf);
```

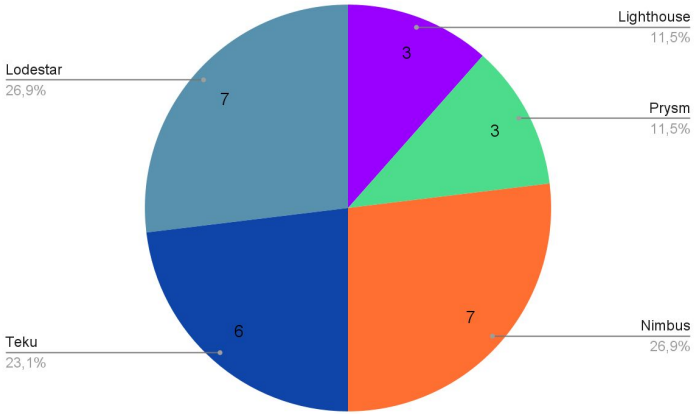
```
var discv5 = require("@chainsafe/discv5");

buf = Buffer.from('656e723a37393639', 'hex').toString()
console.log(buf)

discv5.ENR.decodeTxt(buf);
```

- Limitation
  - Lodestar development was late compared to others
  - Code wasn't always using the **latest specification version**
- Possible improvement
  - Add more fuzzing harnesses to improve coverage

FUZZING LABS

eth2fuzz - Global results

- Total: **26 bugs found**
  - Dumb fuzzing / Replay of coverage-guided generated inputs (10)
  - Coverage-guided fuzzing (16)



- Pros
  - **The most efficient technique** (good ratio bugs/time spent)
  - Coverage-guided fuzzing produce **reusable & interesting corpora**
    - against all targets and by every fuzzing tools
  - All clients have more or less the same naming/function prototype
    - since they **follow the same spec**

- Cons
  - Need to write a lot of fuzzing harnesses
  - Result and speed are dependent on the **fuzzing framework quality/efficiency**
  - Cryptographic BLS signature verification need to be disabled on all targets
    - to speed up fuzzing execution and go deeper into the codebase
  - Difficult to detect **logic bugs**
  - **Time-consuming** to keep up-to-date all fuzzing harnesses

FUZZING LABS

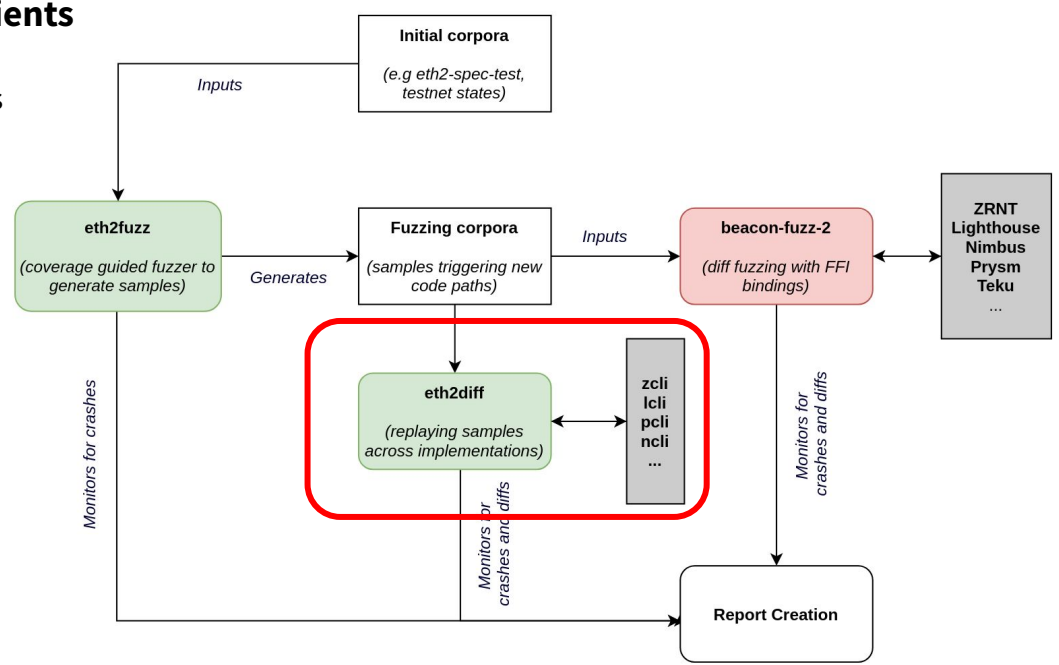
eth2diff - Lazy Differential Fuzzing

eth2diff - (Really) Lazy Differential Fuzzing

- Goals: **Replay fuzzing corpora across clients**
- Complexity: **Low**
  - Compilation of all projects using dockers
  - Extraction of CLI testing tools
    - lci, pcli, ncli, zcli, etc.
  - Comparison of return code
  - Written in Rust

```
// PRYSM
eth2_clients.push(Eth2Client::new(
    "PRYSM".into(),
    cwd.join("shared").join("prysm").join("pcli"),
    [
        "state-transition".into(),
        "--pre-state-path".into(),
        beaconstate,
        "--block-path".into(),
        block,
    ]
    .to_vec(),
));
```

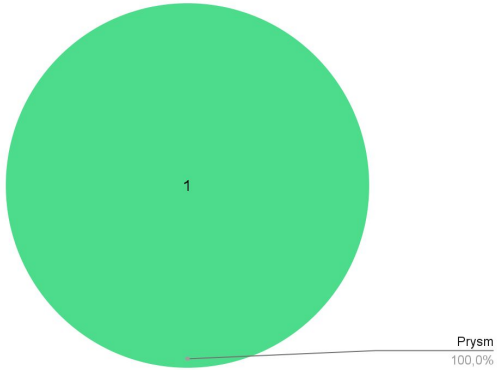
```
// run all eth2clients
process_eth2clients(&mut eth2_clients)?;

// compare the result
compare_results(&eth2_clients)?;
```



eth2diff - Results

- **1 Bug found**
  - PRYSM: Incorrect validation of pre-state attestation & malformed block signature during state transition - link

- Pros
  - **Useful for debugging** crashes found using eth2fuzz
  - Easy and Fast to implement
  - Integration of future ETH2.0 tools/library easier



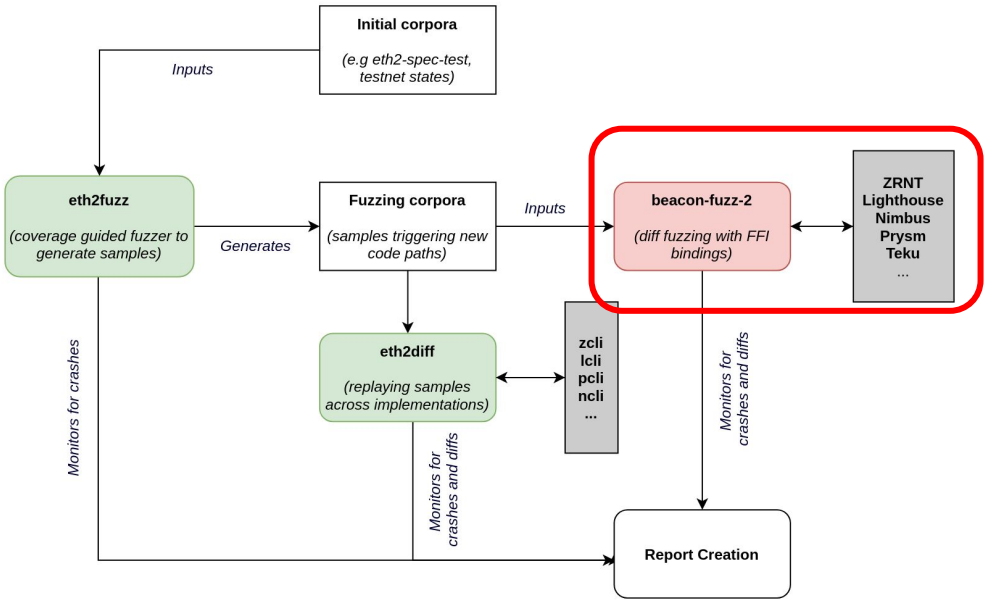
- Cons
  - Really long to compile (> 30 min)
    - 5 projects inside 5 dockers (> 14 Go)
  - Slower than in-process fuzzing
  - **Not a fuzzer, just a differential tester**
    - Don’t generate any inputs, just execute provided ones
  - Not all features (parsing, processing operations) are implemented inside testing CLI tools.
  - Not all projects got working CLI testing tools
  - Some types of inputs are considered as trusted by development teams

beaconfuzz_v2 - Differential fuzzing

beaconfuzz_v2 - Structural & Differential fuzzing

- Goals: **Find logic bugs with differential fuzzing**
  - Reuse existing corpora
  - Generate valid inputs using Structural fuzzing
  - Focus on attacking State processing code
  - Detect outputs difference between all clients

- Complexity: **Medium/Hard**
  - Fuzzing hardnesses in Rust
    - Structural fuzzing
  - FFI Bindings for each client
    - Custom fuzzing library
    - Rust Bindings
  - **A lot of manual writing & compilation**
    - Shared libraries

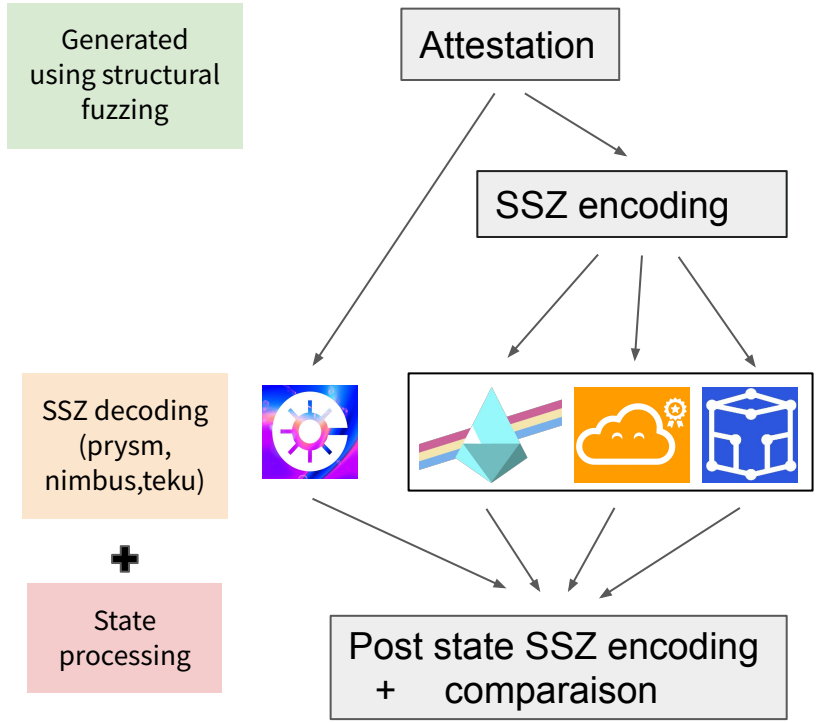


beaconfuzz_v2 - Architecture

- Structural fuzzing in Rust
  - Add Arbitrary trait to each lighthouse structure
    - Create a specific lighthouse branch for fuzzing
    - Dedicated Lighthouse compiler flags

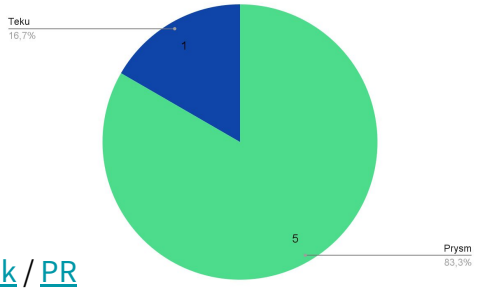
```
#[cfg_attr(feature = "arbitrary-fuzz", derive(arbitrary::Arbitrary))]
```

- **SSZ** encoding/decoding
  - Other clients objects representation are not the same
  - Not possible to just copy memory bytes
- State **processing**
  - Rust FFI Bindings for each client
  - Lighthouse: Direct calls to Rust methods
  - Prysm: Shared fuzzing library (using cgo)
  - Nimbus: Shared fuzzing library
  - Teku: Java Native Interface (JNI) library
- **Comparison**
  - Rust panics if output SSZ objects are differents



beaconfuzz_v2 - Results

- **6 Bugs found (Consensus bugs)**
  - PRYSM: Incorrect Validator Exits verification - link
  - PRYSM: No check of Attestation Indexed Validity - link
  - PRYSM: Incorrect epoch when validating ProposerSlashing - link / PR
  - PRYSM: Invalid verification of attesting indices due to off-by-one bug - link / PR
  - PRYSM: Invalid of proposer slashing when signed block header are equals - link / PR
  - TEKU: Equality of proposer slashing signed block header messages not checked - link / PR



- Pros
  - Structural fuzzing only produce **valid type**
  - SSZ format helps to share data & comparison
  - **Good results finding logic/consensus bugs**
- Cons
  - Writing fuzzing library for each was **really long**
    - Difficult to maintain up-to-date
    - Heavy compilation parts/issues
  - **Slow fuzzing speed**
    - Lot of shared libraries code to execute
    - SSZ decoding/encoding + processing operations

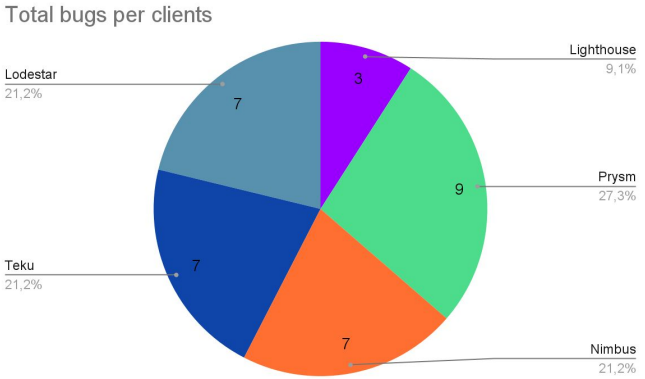
```
#19     INITED cov: 1268 ft: 1376 corp: 17/10736b exec/s: 1 rss: 231Mb
#32     pulse  cov: 1268 ft: 1376 corp: 17/10736b lim: 1240 exec/s: 1 rss: 231Mb
#64     pulse  cov: 1268 ft: 1376 corp: 17/10736b lim: 1240 exec/s: 3 rss: 232Mb
#128    pulse  cov: 1268 ft: 1376 corp: 17/10736b lim: 1240 exec/s: 6 rss: 373Mb
#256    pulse  cov: 1268 ft: 1376 corp: 17/10736b lim: 1240 exec/s: 12 rss: 374Mb
#512    pulse  cov: 1268 ft: 1376 corp: 17/10736b lim: 1240 exec/s: 22 rss: 375Mb
#1024   pulse  cov: 1268 ft: 1376 corp: 17/10736b lim: 1250 exec/s: 37 rss: 375Mb
#2048   pulse  cov: 1268 ft: 1376 corp: 17/10736b lim: 1260 exec/s: 53 rss: 376Mb
#2153   NEW    cov: 1269 ft: 1378 corp: 18/10742b lim: 1260 exec/s: 53 rss: 376Mb L:
#2464   REDUCE cov: 1269 ft: 1378 corp: 18/10739b lim: 1260 exec/s: 57 rss: 377Mb L:
#2681   REDUCE cov: 1269 ft: 1378 corp: 18/10738b lim: 1260 exec/s: 59 rss: 377Mb L:
#2779   REDUCE cov: 1269 ft: 1378 corp: 18/10737b lim: 1260 exec/s: 60 rss: 377Mb L:
#4096   pulse  cov: 1269 ft: 1378 corp: 18/10737b lim: 1270 exec/s: 65 rss: 377Mb
#8192   pulse  cov: 1269 ft: 1378 corp: 18/10737b lim: 1310 exec/s: 1 rss: 377Mb
```

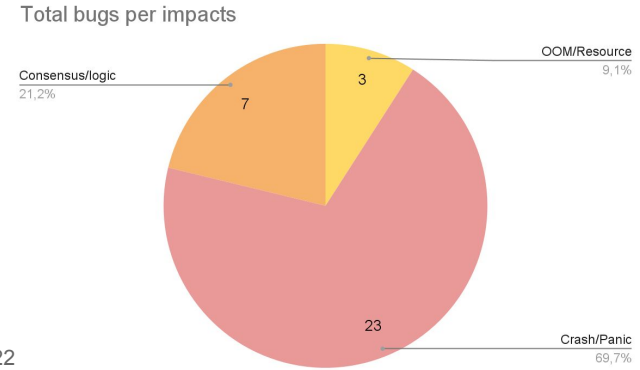
Conclusion

Conclusion & Final results

- Results
  - **33 bugs found (almost all critical)**
    - Lighthouse: 3, Prysm: 9, Nimbus: 7
    - Teku: 7, Lodestar: 7
  - All kinds of bugs found
    - 7 consensus bugs, 23 crashes, 3 OOM/Resource exhaustion
- Community fuzzing
  - Beaconfuzz has been released in Open-source with dockerize version
  - 4 bugs found by other people running beaconfuzz

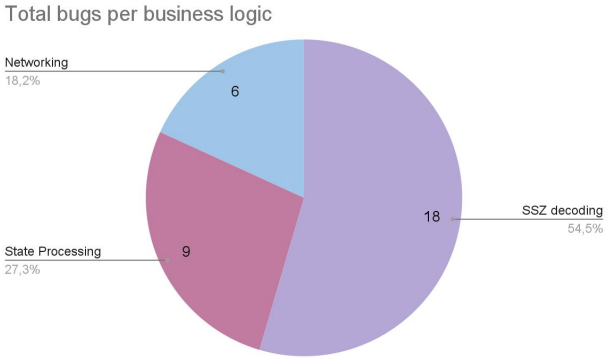
- Main difficulty
  - Complex to keep everything up-to-date
    - Multiple breaking changes
      - 70% of the time spent dealing with compilation issues
    - The specification version supported by each client is different

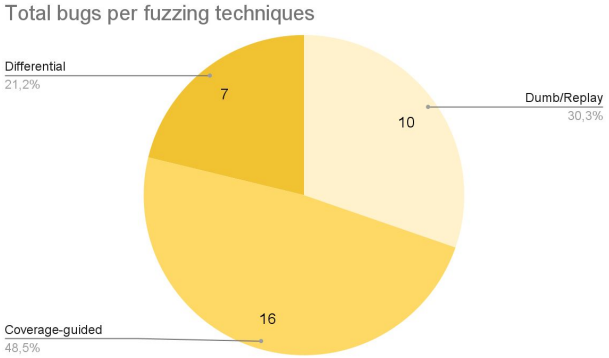




Takeaways & Future

- Takeaways
  - Blockchain software is a **really interesting target**
  - You don't need to build complex fuzzers to start finding bugs
    - 10 bugs with replay, 16 with coverage-guided & 7 differential
    - **Build multiple tools** during your research!
    - Improve them to find more bugs
  - **Differential fuzzing is extremely powerful to find logic bugs**
    - but not discussed publicly a lot.
  - Complete details of this project in the Beaconfuzz series (10 blog posts)
- Future / Next steps
  - Change/Replace some fuzzing framework
    - Especially Jazzer for Teku **(already in production)**
  - Improve & add new fuzzing harnesses
    - Some parts of the code are fuzzed but not with differential testing
    - Networking P2P stack, Lodestar, etc…
  - **Update fuzzing harnesses** for next ETH2.0 Phases
    - Snappy encoding added to SSZ
    - ETH2 specification changed a bit





Thanks for your time! Any questions?

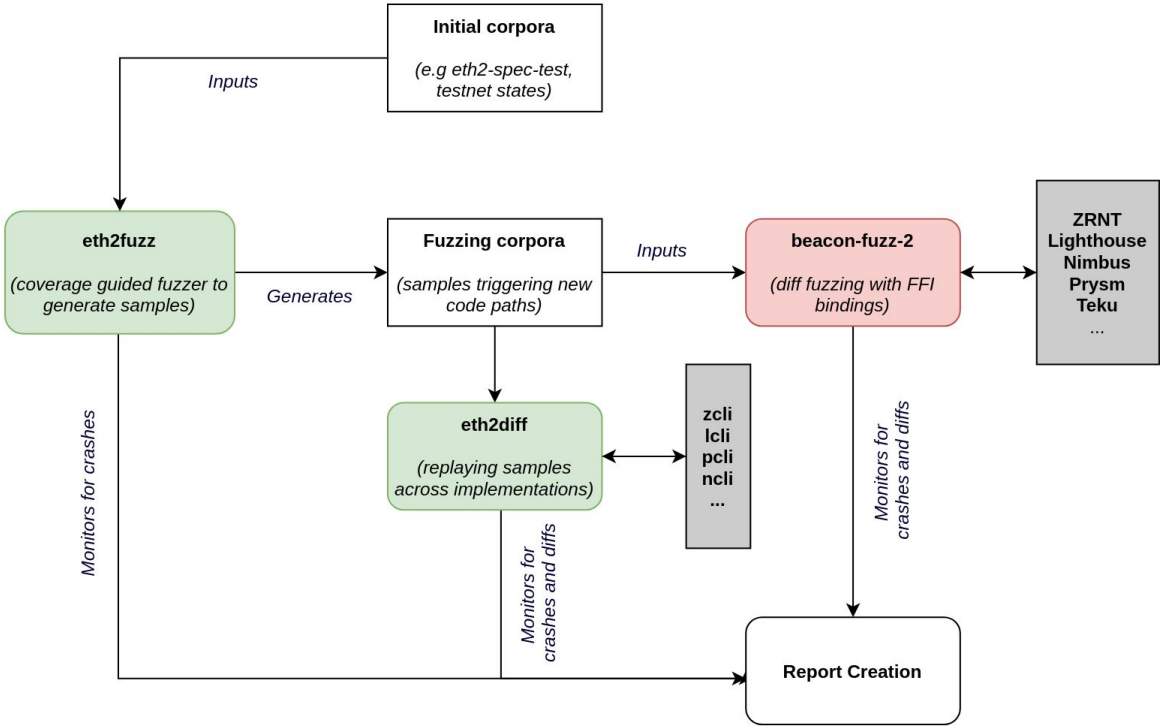


Image sources

- https://pxhere.com/en/photo/636182
- https://ethereum.org/en/assets/
- https://bitcoinmatin.fr/2020/12/02/ethereum-2-0-beacon-lancee/
- https://blog.ethereum.org/2022/01/24/the-great-eth2-renaming/
- https://journalducoin.com/ethereum/ethereum-2-0-dilemme-diversification-clients/
- https://subscription.packtpub.com/book/data/9781839213199/16/ch16lvl1sec17/architecture
- https://github.com/sigp/beacon-fuzz