TRAIL OF BITS

Smart Contracts: The Beta

Hey, I’m Nat 👋

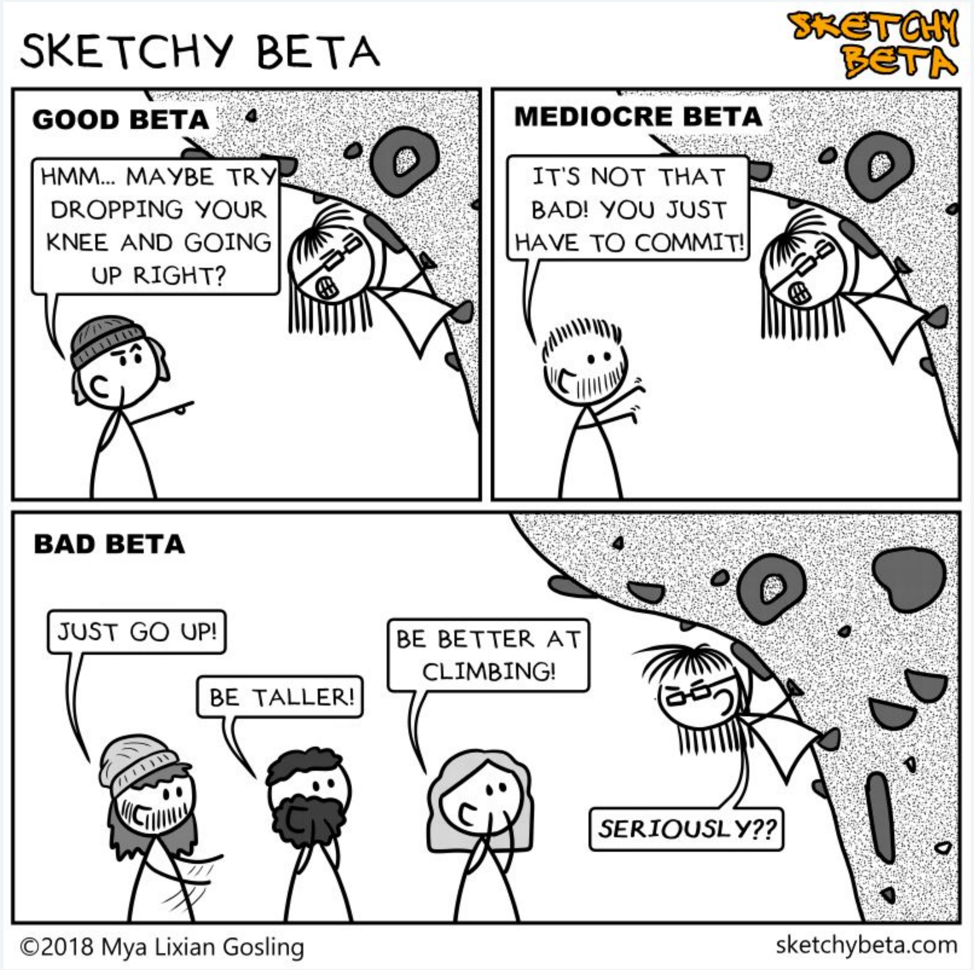
- **Trail of Bits**
- **@0xicingdeath on Twitter**

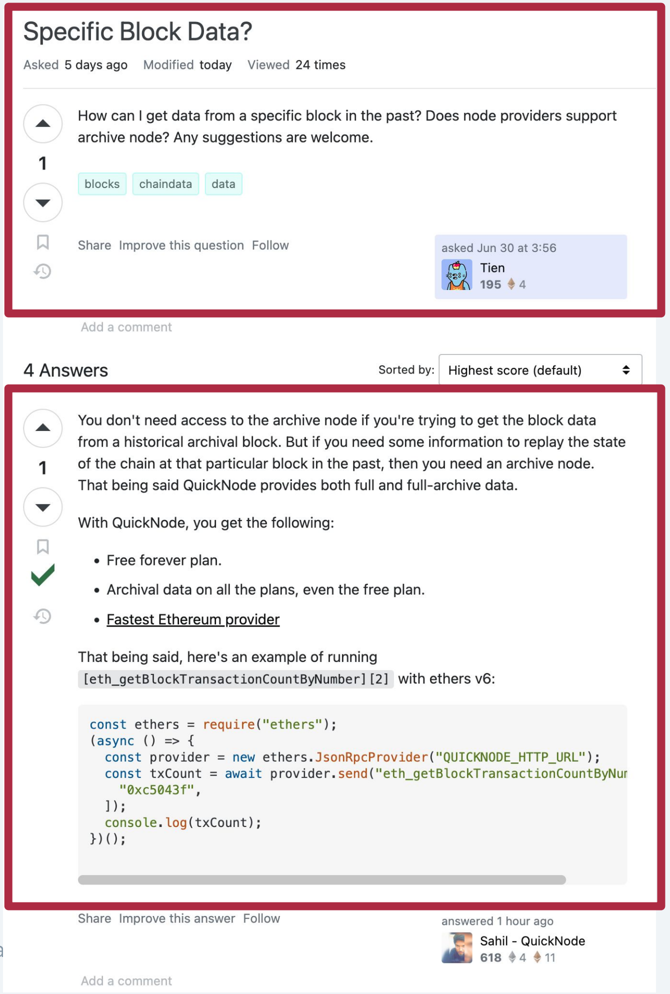
- **tldr work: I break things***
- **I climb things and fall a lot**

Trail of Bits

- We help people build safer software
- Our R&D feeds into the creation of our tools
- And our tools are all open sourced
- Slither, Echidna, Medusa, Caracal, solc-select

Find the rest of our blockchain team here →


SKETCHY BETA
SKETCHY BETA
GOOD BETA
HMM... MAYBE TRY DROPPING YOUR KNEE AND GOING UP RIGHT?
MEDIOCRE BETA
IT'S NOT THAT BAD! YOU JUST HAVE TO COMMIT!
BAD BETA
JUST GO UP!
BE TALLER!
BE BETTER AT CLIMBING!
SERIOUSLY??
©2018 Mya Lixian Gosling
sketchybeta.com



Specific Block Data?

Asked 5 days ago Modified today Viewed 24 times

1

How can I get data from a specific block in the past? Does node providers support archive node? Any suggestions are welcome.

blocks chaindata data

Share Improve this question Follow

asked Jun 30 at 3:56
Tien
195 4

Add a comment

4 Answers Sorted by: Highest score (default)

1

You don't need access to the archive node if you're trying to get the block data from a historical archival block. But if you need some information to replay the state of the chain at that particular block in the past, then you need an archive node. That being said QuickNode provides both full and full-archive data.

With QuickNode, you get the following:

- Free forever plan.
- Archival data on all the plans, even the free plan.
- Fastest Ethereum provider

That being said, here's an example of running `[eth_getBlockTransactionCountByNumber][2]` with ethers v6:

```
const ethers = require("ethers");
(async () => {
  const provider = new ethers.JsonRpcProvider("QUICKNODE_HTTP_URL");
  const txCount = await provider.send("eth_getBlockTransactionCountByNur
    "0xc5043f",
  ]);
  console.log(txCount);
})();
```

Share Improve this answer Follow

answered 1 hour ago
Sahil - QuickNode
618 4 11

Add a comment

The Problem

The Beta

HOW TO WRITE SAFE CODE

4 Answers

Sorted by: Highest score (default)

Let’s find out!

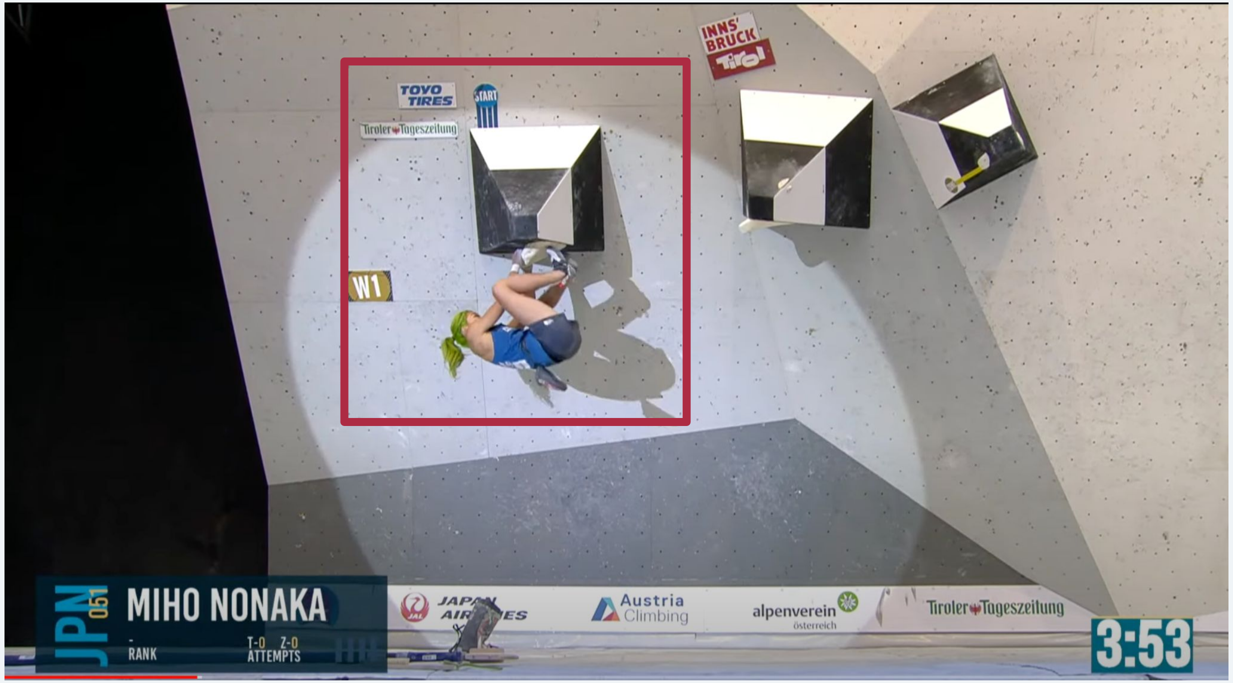
The Problem

The Beta

STEP 1: THE START HOLD

The Starting Hold

The Start



Specifications

- Define **what** you're building
- Define **who** will use your code
- Define **how** it is meant to be used

A good specification includes:

- **Writing down all of your assumptions**
  - Ensure these assumptions hold
- **Identifying and listing all edge cases**
- **Proving all mathematical assumptions**
- **Identifying user flows**
- **Determining data validation assumptions**

A good specification includes:

- **Detailing when your code reverts**
- **Specifying when your code does not revert**
- **Documenting Natspec on all functions**
- **Creating user guides outlining expected behavior**

Benefits of specifications

- You'll **know** what your code is supposed to do
- You'll **know** how your code is supposed to work

How to write a spec

- **Ensure your spec is up to date**
- **Check it into source control**
- **Update it every time you change code**

Step 2: USE THE ~~FORCE~~ TOOLS

Climbing Safely

- We use **tools** to stay safe
- Rope, carabiners, anchors, cams
- Anchors mounted throughout large walls protect us

📸 by @cheynelempe

Tooling in blockchain security

Your Anchor Point: Unit Testing

You wouldn’t want your anchor to look like this

So what *should* you do?

- **Use your detailed specifications**
- **Test “Happy paths” (expected features)**
- **Test “Unhappy path” (expected failure features)**
- **Identify your edge cases early**
- **Minimum 100% statement and branch coverage**

Benefits of Unit Testing

- **Find deviations between specifications and code easily**
- **Forces you to write testable code**
- **Identify bugs as early as possible while developing**
- **Future-proofs your code from re-introducing the same bug**

Limitations of Unit Testing

- **Tests are only as good as your inputs**
- **Does not test integration between components**

Considering Edge Cases

- **Identify safe bounds of input**
- **What does your code do under unsafe bounds?**
- **Consider what your code does not handle**
- **We’ll come back to this ;)**

CLIMBING HOLDS
SKETCHY BETA
#013
JUG
SLOPER
PINCH
POCKET
CRIMP
CREDIT CARD CRIMP
THAT'S... THAT'S GOT TO BE A FOOT CHIP, RIGHT?
...WTF YOU HAVE GOT TO BE KIDDING ME
©2018 Mya Lixian Gosling
@SketchyBeta / sketchybeta.com

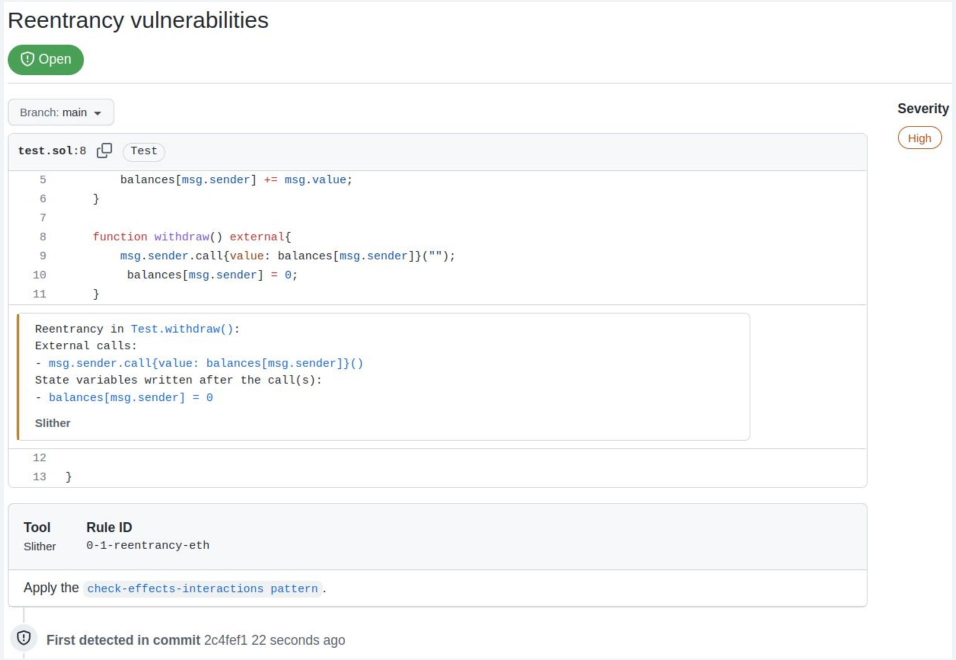
A Jug: Static Analysis

- **We love them**
- **They make our climb easier**
- **They're like monkey bars**

Static Analysis:

Your **automatic** bug finder

Static Analysis: An Example



Reentrancy vulnerabilities

Open

Branch: main

Severity

High

test.sol:8 Test

```
        balances[msg.sender] += msg.value;
    }

    function withdraw() external{
        msg.sender.call{value: balances[msg.sender]}("");
         balances[msg.sender] = 0;
    }
```

```
Reentrancy in Test.withdraw():
External calls:
- msg.sender.call{value: balances[msg.sender]}()
State variables written after the call(s):
- balances[msg.sender] = 0
```

Slither

```

}
```

Tool	Rule ID
Slither	0-1-reentrancy-eth

Apply the `check-effects-interactions pattern`.

First detected in commit 2c4fef1 22 seconds ago

Static Analysis: Slither

https://github.com/crytic/slither

- Will tell you what **might** be problematic
- Analyzes your code **statically**
- Matches **87 detectors** with commonly known vulnerabilities

Slither Findings: What do?

- **If code is currently exploitable, fix it**
- **If code can be exploitable, document it**
- **If code is not currently exploitable, silent** Slither and **document it**

Pros and Cons: Static Analysis

- **Runs quickly**
- **Tells you what looks wrong**
- **Enforces best practices**

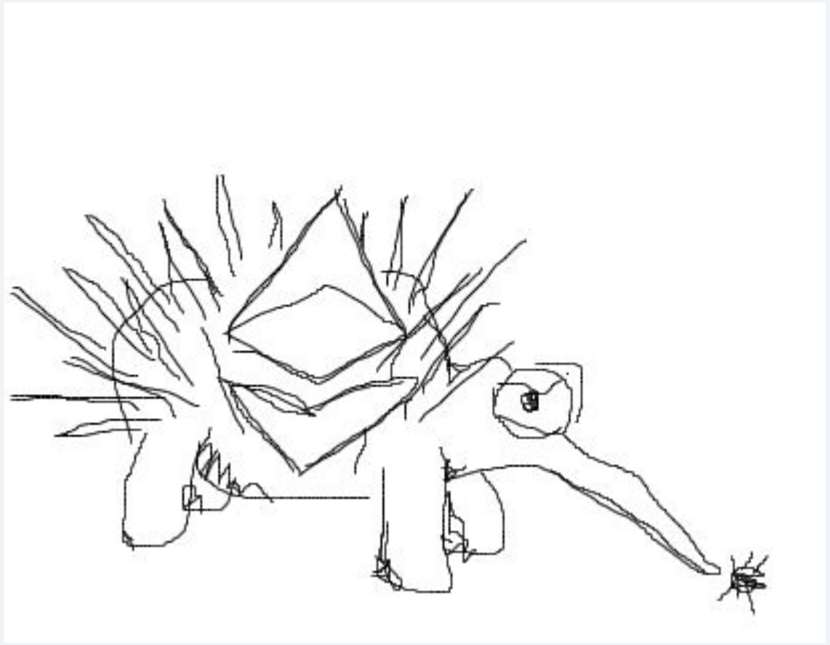
- **Verbose, high false positive rate**

The Crux: Fuzzing

Fuzzing

- Stress your code with random inputs
- Forces you to **identify system properties (invariants)**
- Provides **roboticized unit testing**
- Can be more challenging to set up E2E

Echidna

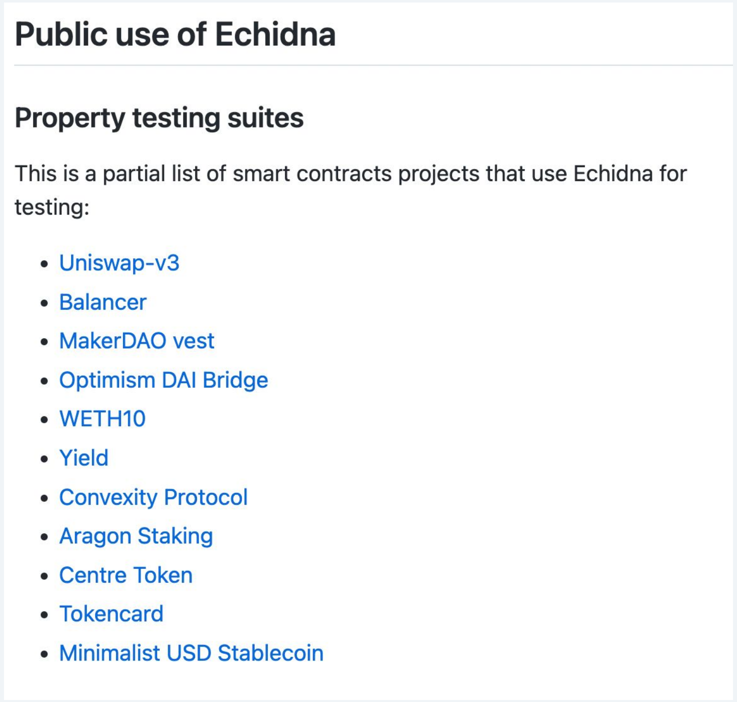


Echidna

https://github.com/crytic/echidna

- **Smart contract fuzzer**
- **Automatically generates inputs for functions**
- **Detects and highlights property violations**

Watch for developments on Medusa


Public use of Echidna

Property testing suites

This is a partial list of smart contracts projects that use Echidna for testing:

- Uniswap-v3
- Balancer
- MakerDAO vest
- Optimism DAI Bridge
- WETH10
- Yield
- Convexity Protocol
- Aragon Staking
- Centre Token
- Tokencard
- Minimalist USD Stablecoin

Pros and Cons of Fuzzing

- Finds **complex** bugs
- Detect edge cases
- More complete than unit testing

- Can be harder to setup

SAFETY CHECKS
SKETCHY BETA #092
HARNESS DOUBLED BACK?
HARNESS DOUBLED BACK!
KNOT CHECK?
TWO FOUR SIX EIGHT TEN! LOOKS GOOD!
CLIMBER'S SIDE ON TOP, GRIGRI CAMS, CARABINER LOCKED, AND WE'RE ON THE SAME ROPE! YOU'RE GOOD TO GO!
CLIMBING!
CLIMB ON!
I CHANGED MY MIND. I DON'T WANT TO CLIMB THIS ONE.
©2020 Mya Lixian Gosling
@SketchyBeta / sketchybeta.com

Blockchain Security Safety Checks

- Security is a **continuous process**
- Write **everything** down
- **Add unit tests as you write new code**
- **Run Slither on every new commit**
- **Determine invariants**
- **Run fuzz tests**

Want more beta?

https://secure-contracts.com/

- Incident Response Plan
- Development Workflow
- Risks related to token integrations
- EVM and EVM Opcodes
- Transaction Tracing
- + way more