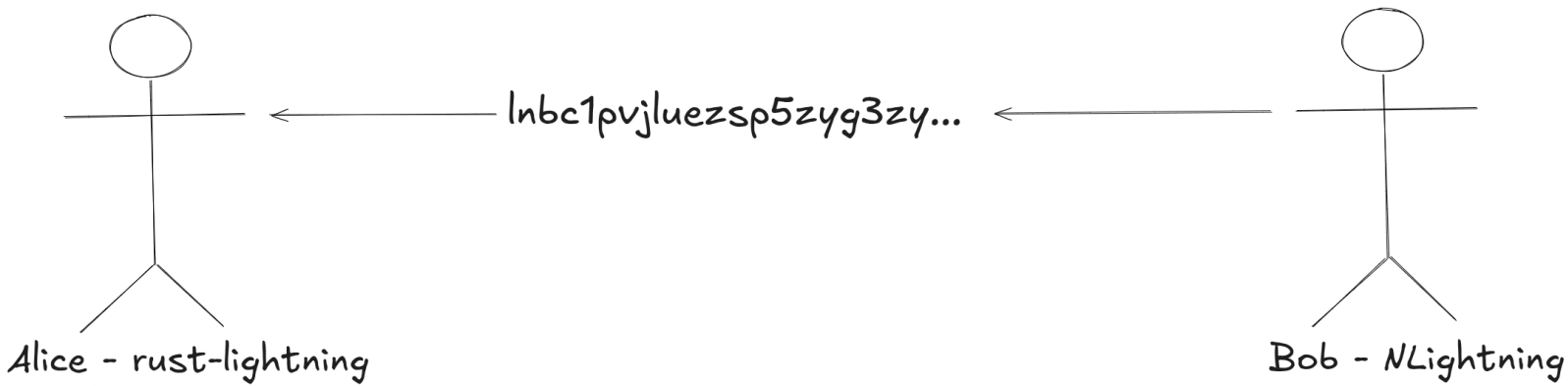
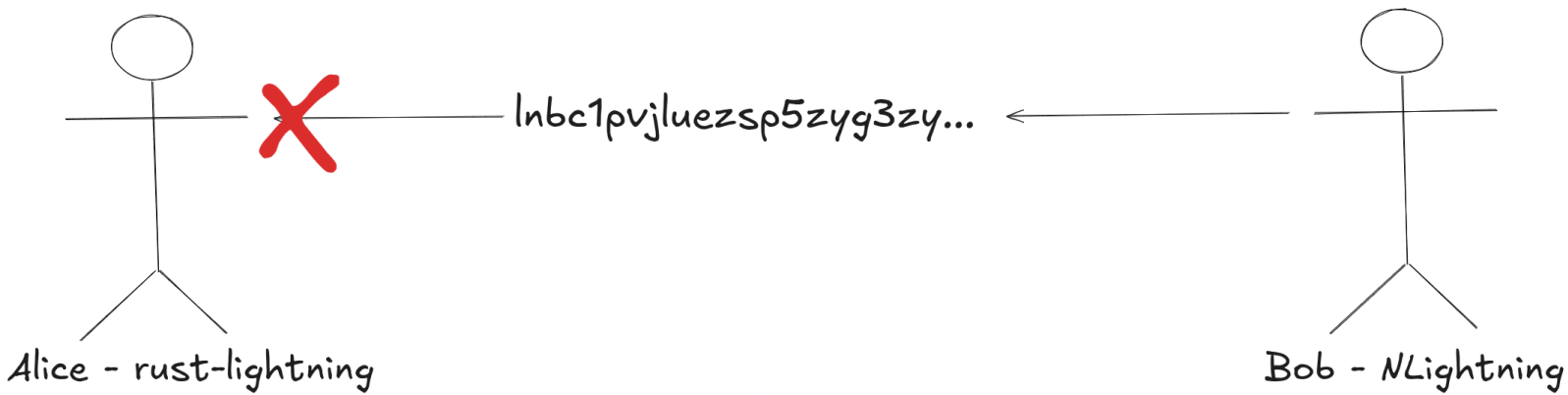
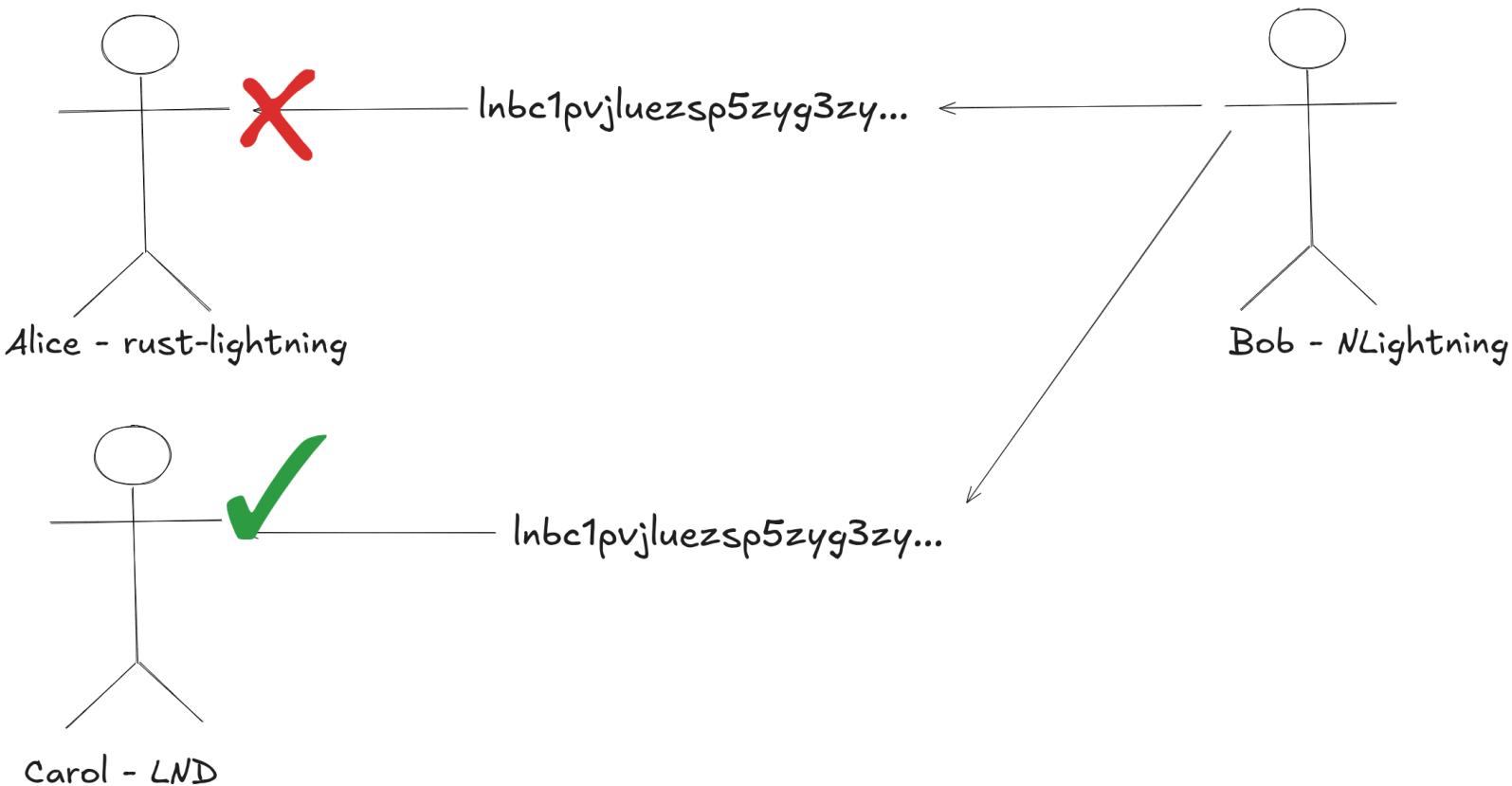
Differential Fuzzing on ⚡


lnbc1pvjluezsp5zyg3zy...
Alice - rust-lightning
Bob - NLightning


lnbc1pvjluezsp5zyg3zy...
Alice - rust-lightning
Bob - NLightning


lnbc1pvjluezsp5zyg3zy...
Alice - rust-lightning
Bob - NLightning
lnbc1pvjluezsp5zyg3zy...
Carol - LND

Same Spec, Different Behavior

The problem:

- Network Instability
- Bad User Experience
- Potential Security Issues

Traditional approach:

- Wait for users to report bugs
- Manual cross-implementation testing
- Reactive fixes after failures

How do we systematically find these discrepancies before they cause problems?

Solution: Differential Fuzzing

Who am I?

- Erick Cestari
- Vinteum Grantee (Bitcoin development funding)
- Contributor of Bitcoinfuzz. Found 15+ bugs across Lightning implementations (we'll see some of them)

Bitcoin: Code as Specification

If we built a new Bitcoin implementation today, where would we find the specification?

- Bitcoin Core codebase. The reference implementation
- No formal written specification document
- Consensus rules are implicit in the code

Lightning: Specification-First Approach

Lightning Network took a different approach with BOLT specifications

- BOLT = Basis of Lightning Technology
- Formal written specifications for all protocol aspects
- Multiple implementations can follow the same spec
- But... specifications can be ambiguous or incomplete

Edge Cases

When the spec says `r field should contain one or more entries`. What happens with zero entries?

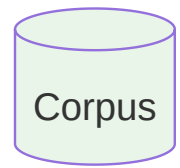
This is where implementations diverge:

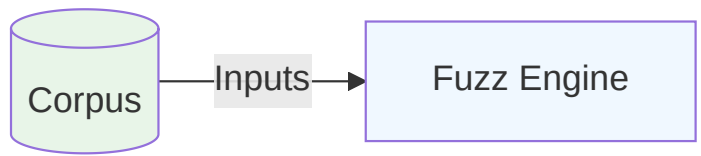
- Some reject it (Rust-Lightning, Core Lightning)
- Others accept it (LND, Eclair)

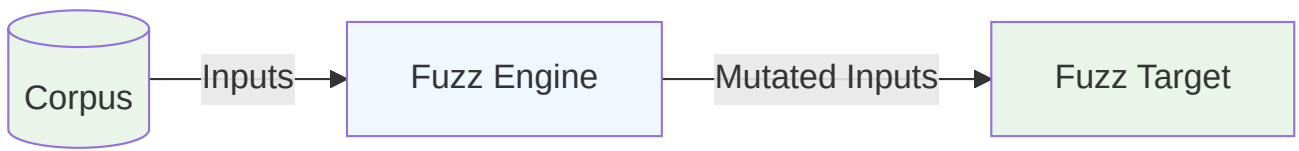
Differential fuzzing systematically explores these specification gaps.

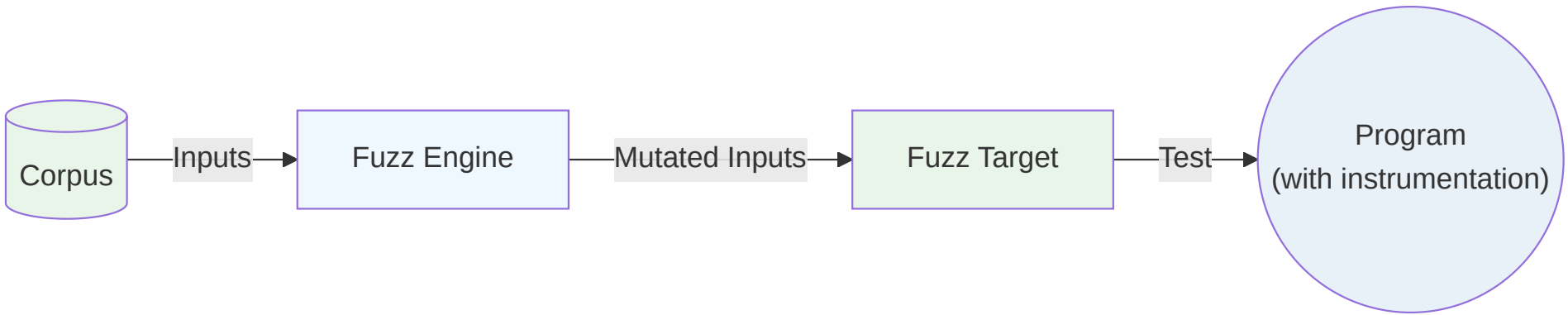
So let’s start simple, what is fuzzing?

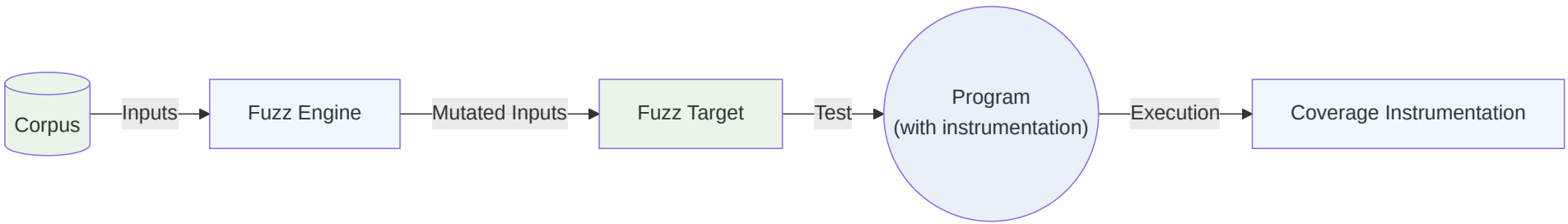
Fuzzing is an automated software testing technique that involves providing invalid, unexpected, or random data as inputs to a computer program.

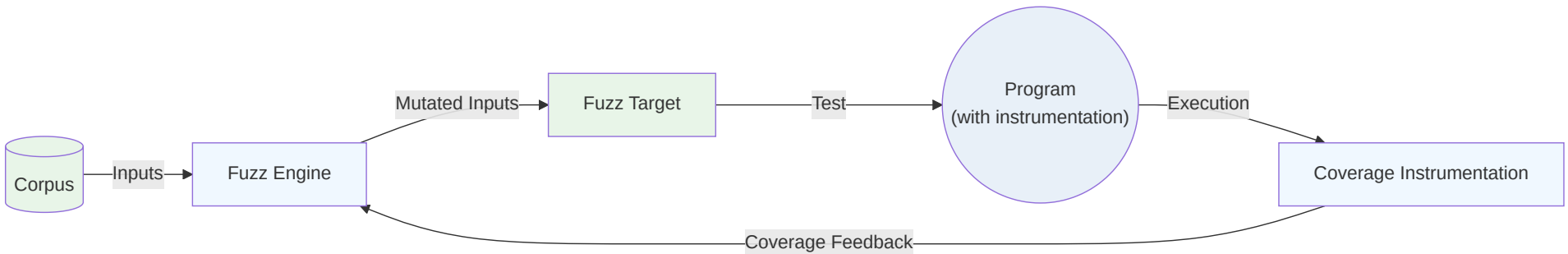

Corpus

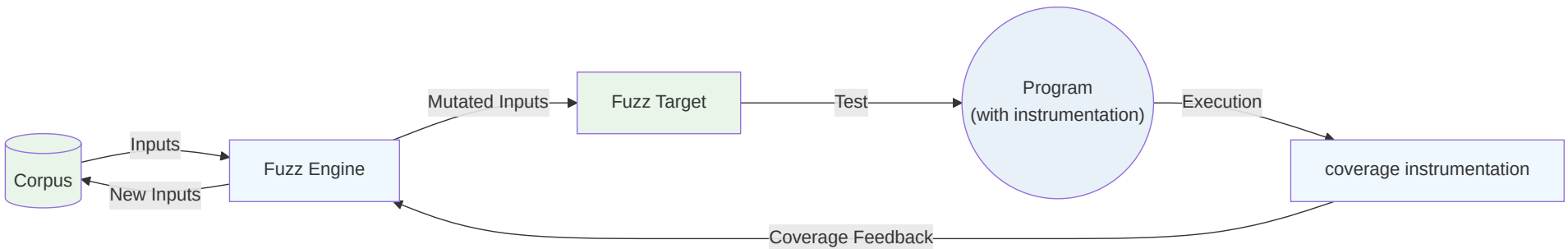

Corpus
Inputs
Fuzz Engine


Corpus
Inputs
Fuzz Engine
Mutated Inputs
Fuzz Target


Corpus
Inputs
Fuzz Engine
Mutated Inputs
Fuzz Target
Test
Program
(with instrumentation)


Corpus
Inputs
Fuzz Engine
Mutated Inputs
Fuzz Target
Test
Program
(with instrumentation)
Execution
Coverage Instrumentation


Corpus
Inputs
Fuzz Engine
Mutated Inputs
Fuzz Target
Test
Program
(with instrumentation)
Execution
Coverage Instrumentation
Coverage Feedback

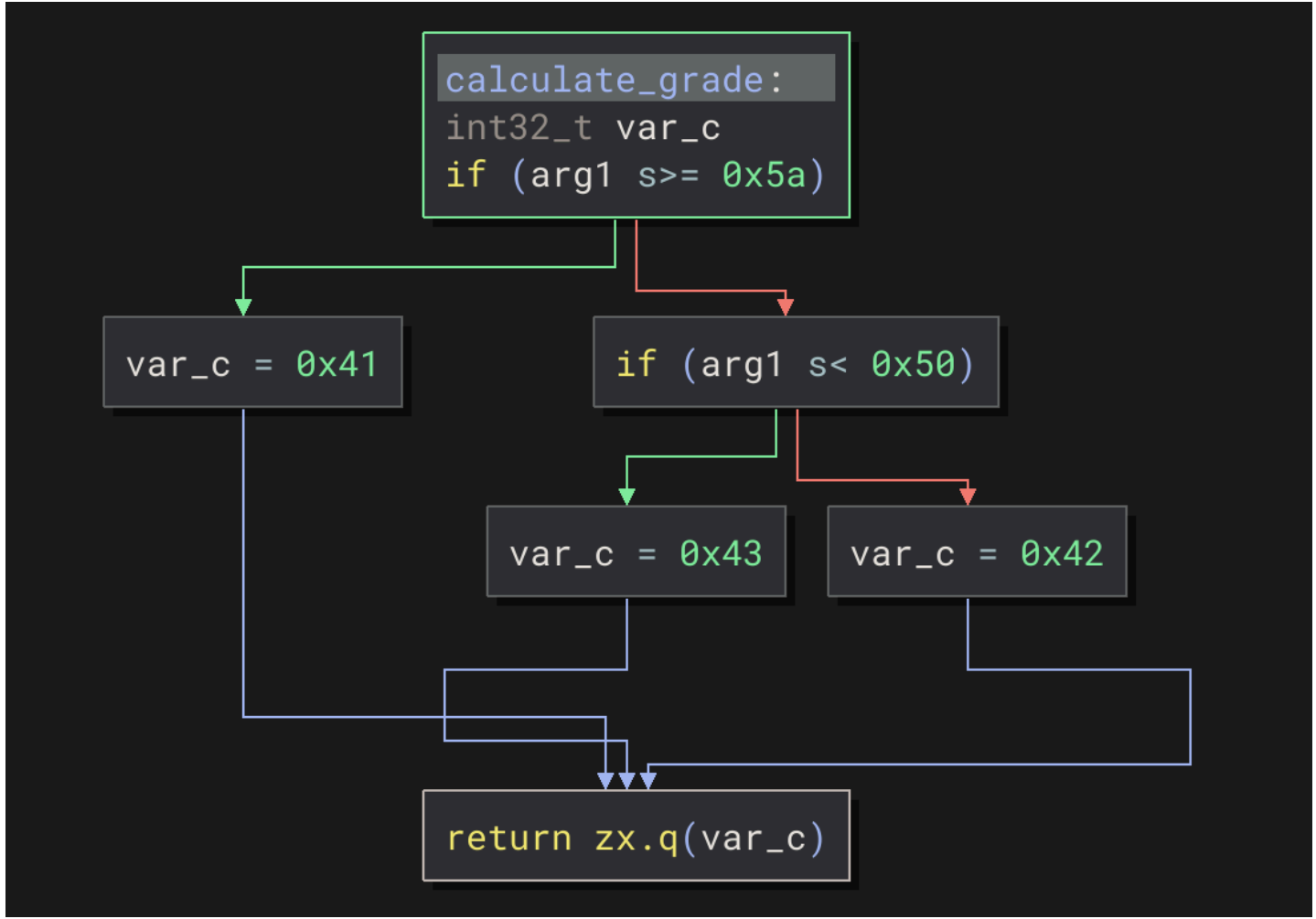

Corpus
Inputs
New Inputs
Fuzz Engine
Mutated Inputs
Fuzz Target
Test
Program
(with instrumentation)
Execution
coverage instrumentation
Coverage Feedback

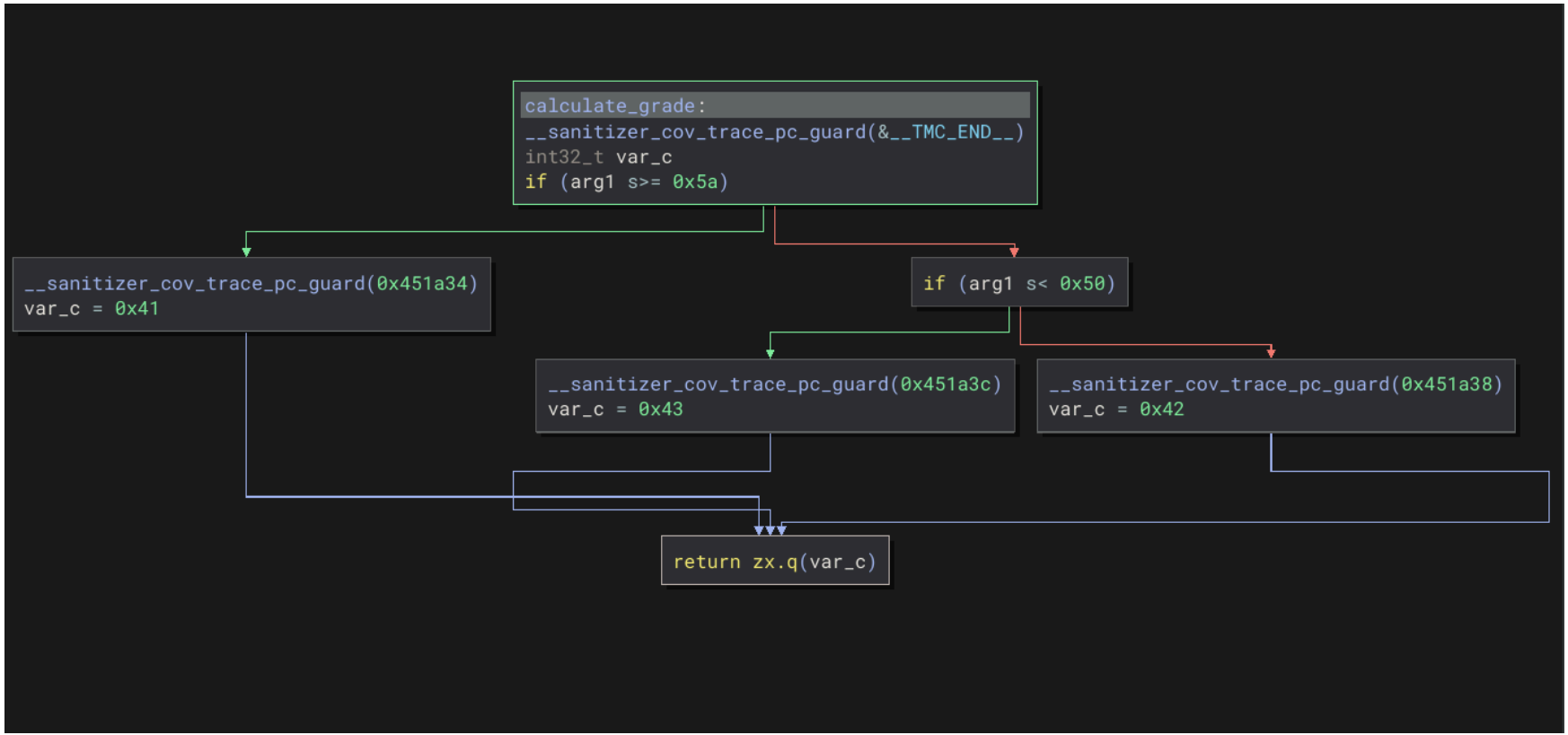
Coverage instrumentation

We use instrumentation to track which code paths are exercised.

It inserts calls to user-defined functions on function-, basic-block-, and edge- levels.

```c
#include <stdio.h>
#include <stdlib.h>

int calculate_grade(int score) {
    if (score ⩾ 90) {
        return 'A';
    } else if (score ⩾ 80) {
        return 'B';
    } else {
        return 'C';
    }
}
```


calculate_grade:
int32_t var_c
if (arg1 s>= 0x5a)
var_c = 0x41
if (arg1 s< 0x50)
var_c = 0x43
var_c = 0x42
return zx.q(var_c)


calculate_grade:
__sanitizer_cov_trace_pc_guard(&__TMC_END__)
int32_t var_c
if (arg1 s>= 0x5a)
__sanitizer_cov_trace_pc_guard(0x451a34)
var_c = 0x41
if (arg1 s< 0x50)
__sanitizer_cov_trace_pc_guard(0x451a3c)
var_c = 0x43
__sanitizer_cov_trace_pc_guard(0x451a38)
var_c = 0x42
return zx.q(var_c)

Fuzzing Finds Crashes

What is the problem with this double function?

```
use libfuzzer_sys::fuzz_target;

fn double(x: i32) -> i32 {
    x * 2
}

fuzz_target!(|data: &[u8]| {
    if let Some(x) = consume_i32(data) {
        let _ = double(x);
    }
});
```

When Fuzzing Gets Stuck

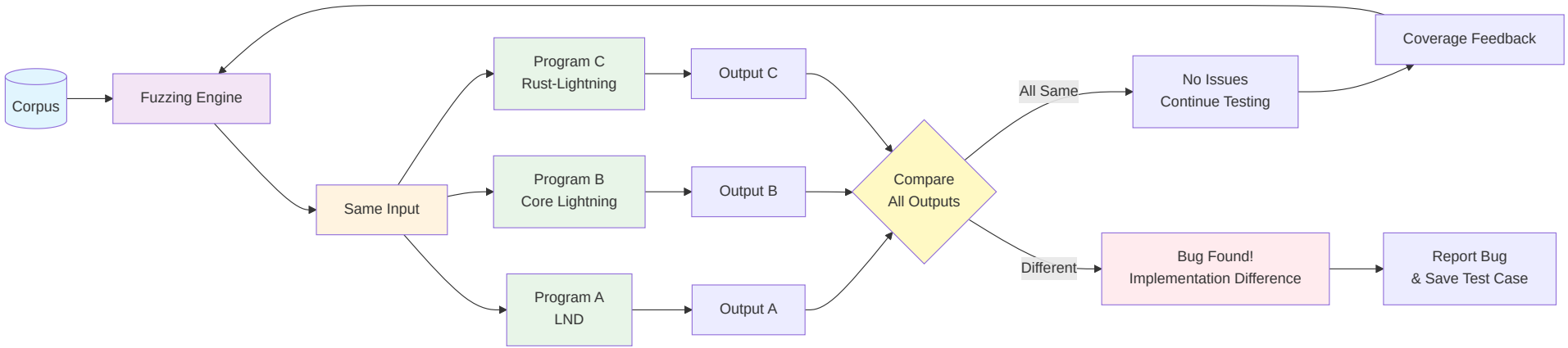
```rust
use libfuzzer_sys::fuzz_target;

fn double(x: i32) -> Option<i32> {
    x.checked_mul(2)
}

fuzz_target!(|data: &[u8]| {
    if let Some(x) = consume_i32(data) {
        let _ = double(x);
    }
});
```

Coverage-Guided Fuzzing in Action

```rust
use libfuzzer_sys::fuzz_target;

fuzz_target!(|data: &[u8]| {
    if data.len() != 6 {
        return;
    }

    if data[0] == b'f' {
        if data[1] == b'u' {
            if data[2] == b'z' {
                if data[3] == b'z' {
                    if data[4] == b'l' {
                        if data[5] == b'n' {
                            panic!("Crash triggered: found the magic sequence!");
                        }
                    }
                }
            }
        }
    }
});
```

Differential Fuzzing

- Generate inputs and feed them simultaneously to **multiple programs**.
- Compare the outputs of the programs to find discrepancies.


Corpus
Fuzzing Engine
Same Input
Program C
Rust-Lightning
Program B
Core Lightning
Program A
LND
Output C
Output B
Output A
Compare
All Outputs
All Same
Different
No Issues
Continue Testing
Bug Found!
Implementation Difference
Report Bug
& Save Test Case
Coverage Feedback

Example: Differential Fuzzing

```
fn double(x: i32) → Option<i32> {
    x.checked_mul(2)
}

fn double2(x: i32) → Option<i32> {
    // Off-by-one: using ⩾ instead of >
    if x ⩾ i32::MAX / 2 || x < i32::MIN / 2 {
        None
    } else {
        Some(x * 2)
    }
}

fuzz_target!(|data: &[u8]| {
    if let Some(x) = consume_i32(data) {
        let res = double(x);
        let res2 = double2(x);
        if res ≠ res2 {panic!("x: {}, res: {:?}, res2: {:?}", x, res, res2);}
    }
});
```

Bitcoinfuzz: Bitcoin Differential Fuzzing

What we're building:

- Differential Fuzzing framework for Bitcoin protocol implementations and libraries
- Focus: Find discrepancies before they cause issues

Current targets (for lightning network):

- modules: LND, Core Lightning, Rust-Lightning, Eclair, lightning-kmp
- targets: deserialize_invoice, deserialize_offer, parse_p2p_lightning_message

Status: 30 bugs found so far.

How does Bitcoinfuzz works with different languages?

- Compile with instrumentation (if possible)

LND:

```
go build -o liblnd_wrapper.a -buildmode=c-archive -tags=libfuzzer -gcflags=all=-d=libfuzzer wrapper.go
```

rust-lightning:

```
RUSTFLAGS="-Z sanitizer=address" cargo rustc --release -- -C passes='sancov-module' \
    -C llvm-args=-sanitizer-coverage-inline-8bit-counters \
    -C llvm-args=-sanitizer-coverage-trace-compares \
    -C llvm-args=-sanitizer-coverage-pc-table \
    -C llvm-args=-sanitizer-coverage-level=4 \
    -C llvm-args=-simplifycfg-branch-fold-threshold=0
```

How to fuzz interpreted or VM-based languages?

- Use ffi (foreign function interface)
- For java, use JNI

Eclair:

```
static std::optional<std::string> eclair_decode_invoice(const char* invoiceStr) {
    if (!init_jvm() || !jvm) {
        std::abort();
    }

    JNIEnv* env = nullptr;
    jint status = jvm->GetEnv((void**)&env, JNI_VERSION_1_8);

    jstring jInvoiceStr = env->NewStringUTF(invoiceStr);
    if (!jInvoiceStr) {
        return "";
    }

    jstring jResult = static_cast<jstring>(
        env->CallStaticObjectMethod(decoderClass, decodeMethod, jInvoiceStr)
    );
    env->DeleteLocalRef(jInvoiceStr);
```

Let’s breakdown a target in Bitcoinfuzz

- We need to define at least three things to create a target:

1. Target function
2. Input type
3. Output type
4. Custom mutators (if needed)

Example: deserialize_invoice target

1. Input type: BOLT11 invoice string
2. Output type: string containing all the invoice data (e.g., amount, description, etc.)
3. Custom mutators: Bech32 custom mutator

rust-lightning:

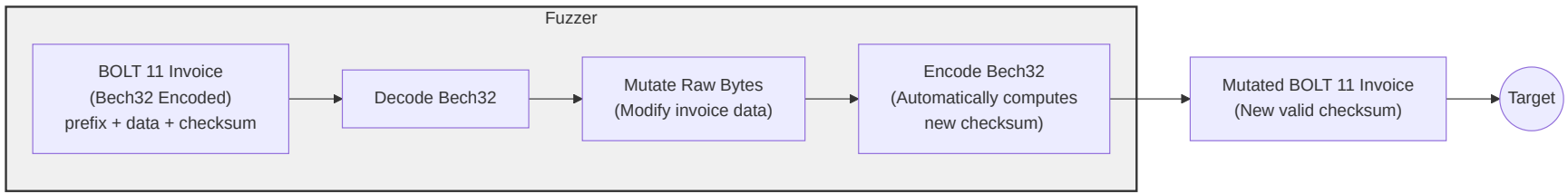
```rust
#[no_mangle]
pub unsafe extern "C" fn ldk_des_invoice(input: *const std::os::raw::c_char) -> *mut c_char {
    if input.is_null() {
        return str_to_c_string("");
    }

    // Convert C string to Rust string
    let c_str = match CStr::from_ptr(input).to_str() {
        Ok(s) => s,
        Err(_) => return str_to_c_string(""),
    };

    match Bolt11Invoice::from_str(c_str) { // <-- target function
        Ok(invoice) => {
            if invoice.currency() != Currency::Bitcoin {
                return str_to_c_string("");
            }
            let mut result = String::new(); // <-- output

            result.push_str("HASH=");
            result.push_str(&invoice.payment_hash().to_string());

            result.push_str(";PAYMENT_SECRET=");
            result.push_str(&invoice.payment_secret().to_string());
            ...
```

LND:

```
//export LndDeserializeInvoice
func LndDeserializeInvoice(cInvoiceStr *C.char) *C.char {
  if cInvoiceStr == nil {
    return C.CString("")
  }

  runtime.GC()

  // Convert C string to Go string
  invoiceStr := C.GoString(cInvoiceStr)

  network := &chaincfg.MainNetParams

  invoice, err := zpay32.Decode(invoiceStr, network) // ⟵ target function
  if err ≠ nil {
    return C.CString("")
  }

  var sb strings.Builder // ⟵ output

  sb.WriteString("HASH=")
  if invoice.PaymentHash ≠ nil {
    sb.WriteString(fmt.Sprintf("%x", *invoice.PaymentHash))
  }
    ...
```

Custom mutator

- BOLT 11 invoice have checksums that are calculated from the data.
- So we need to mutate the data and recompute the checksum.



Compare the outputs

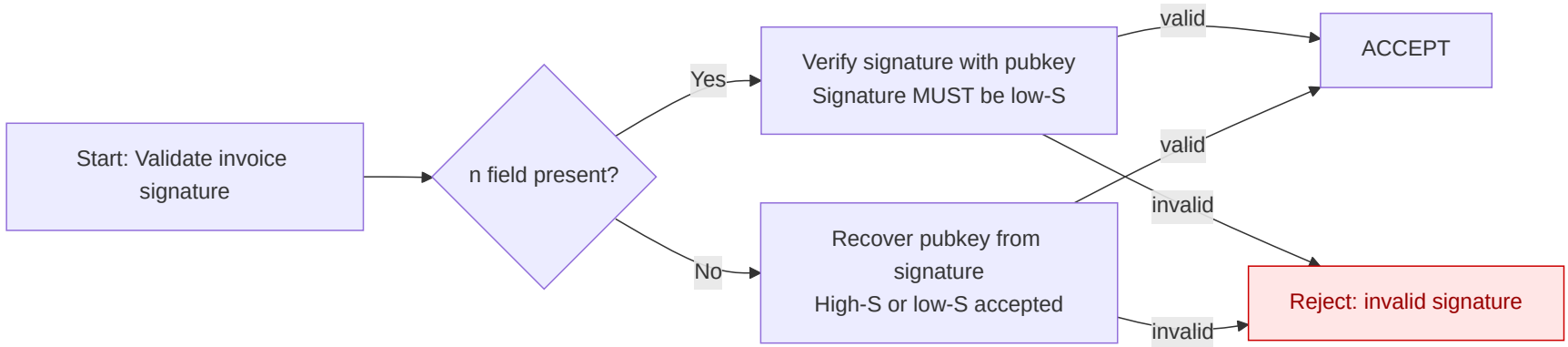
```
void Driver::InvoiceDeserializationTarget(std::span<const uint8_t> buffer) const {
    std::string invoice = ConsumeString(buffer);
    std::optional<std::string> first_result;

    // Test each Lightning implementation
    for (auto &implementation : implementations) {
        auto result = implementation->deserialize_invoice(invoice);

        if (first_result.has_value()) {
            // Compare with previous result
            if (result != first_result) {
                std::cout << "DISCREPANCY FOUND!" << std::endl;
                std::cout << "Invoice: " << invoice << std::endl;
                std::cout << "Implementation A: " << first_result.value() << std::endl;
                std::cout << "Implementation B: " << result.value() << std::endl;

                // This assertion will crash and save the test case
                assert(result == first_result);
            }
        } else {
            first_result = result;
        }
    }
}
```

Let’s run Bitcoinfuzz!

So what bugs have we found so far?

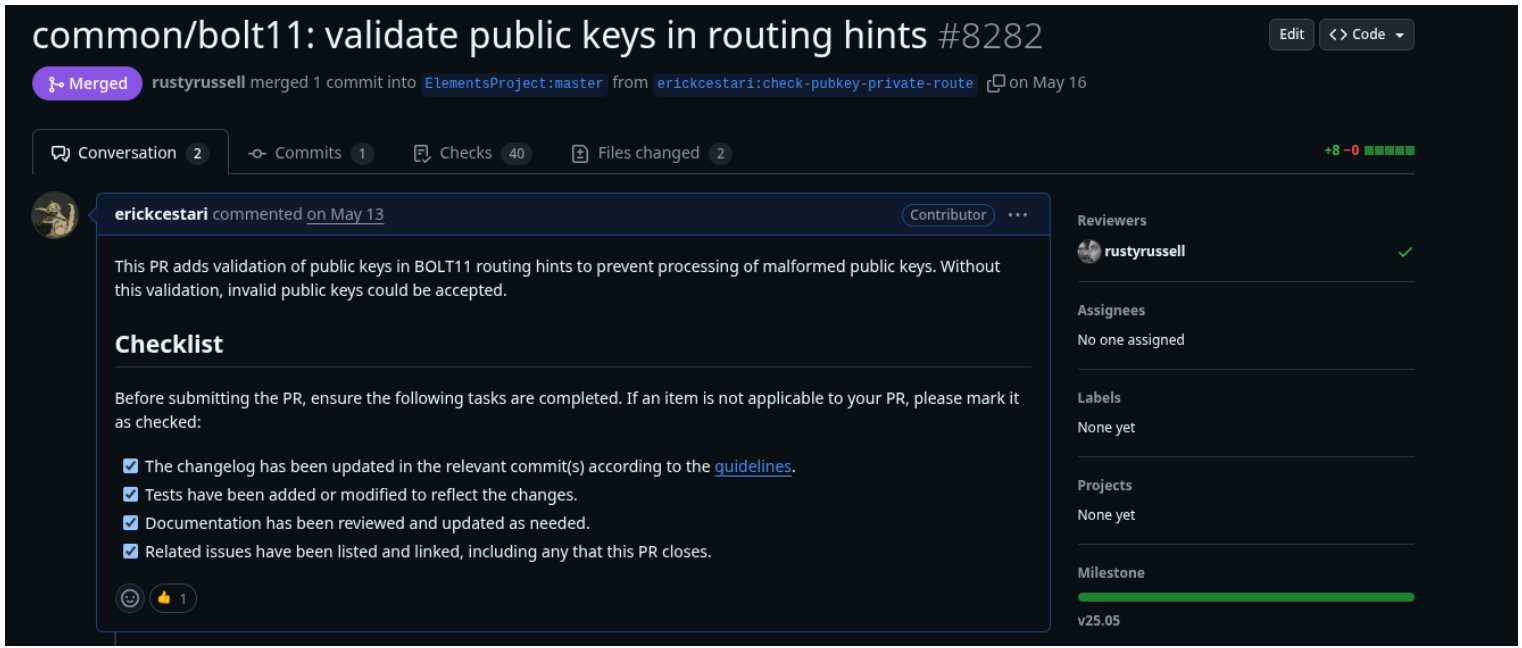
1. LND: lightningnetwork/lnd#9591
2. Core Lightning: ElementsProject/lightning#8219
3. LND: lightningnetwork/lnd#9808
4. Core Lightning: ElementsProject/lightning#8282
5. bolts: lightning/bolts#1264
6. rust-lightning: lightningdevkit/rust-lightning#3814
7. LND: lightningnetwork/lnd#9904
8. LND: lightningnetwork/lnd#9915
9. Eclair: ACINQ/eclair#3104
10. lightning-kmp: ACINQ/lightning-kmp#799
11. lightning-kmp: ACINQ/lightning-kmp#801
12. lightning-kmp: ACINQ/lightning-kmp#802
13. rust-lightning: lightningdevkit/rust-lightning#3998
14. bolts: lightning/bolts#1279
15. rust-lightning: lightningdevkit/rust-lightning#4018

Improved signature specification in BOLT11



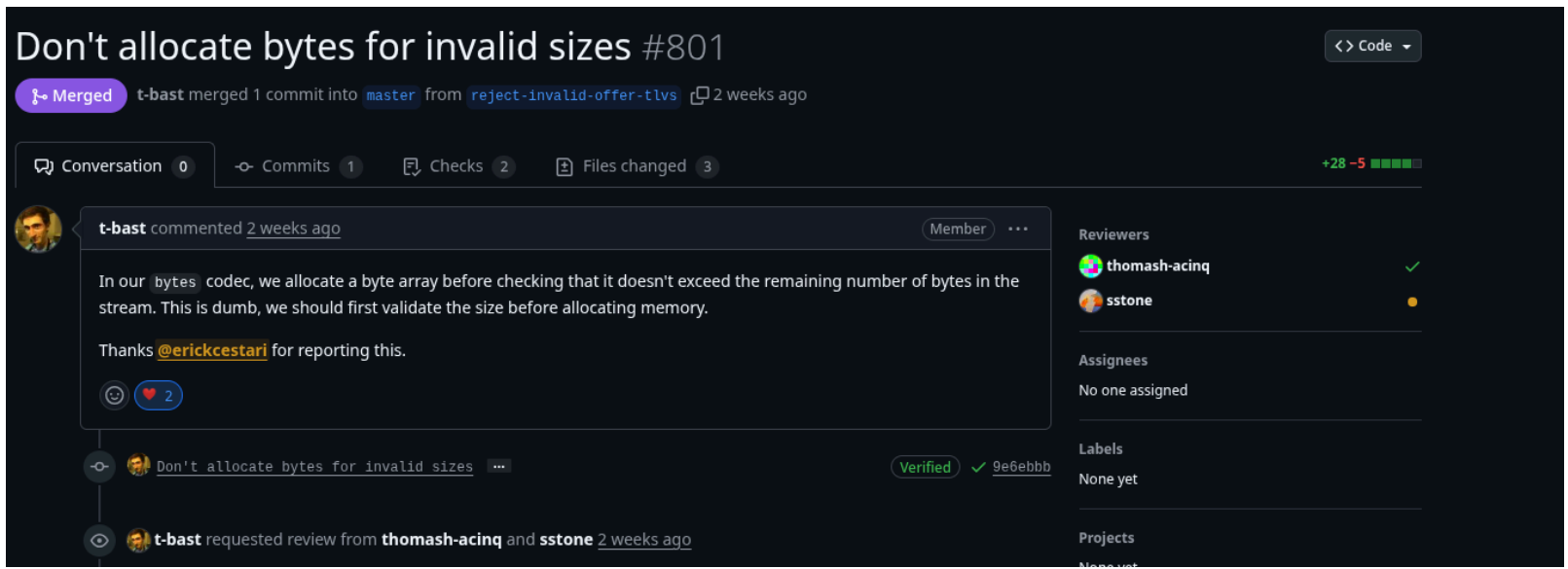
C-Lightning accepting invalid invoices

C-Lightning was accepting invoices with routing hints without validating the public key format.



Doing differential fuzzing of projects that do not have fuzz testing

Some projects do not have support for fuzzing or do not run their fuzz targets continuously. It means that we could find bugs not because of the "differential" thing, but simply because the project has not been fuzzed.



How to Contribute to **Bitcoinfuzz**

- **Add new targets** (BIP32 key derivations, secp256k1, etc.)
- **Add new libraries** (bolt11.js, bitcoinj, libbitcoin)
- **Build system improvements**
- **Run bitcoinfuzz!**

Thank You!

erickcestari03@gmail.com
github.com/erickcestari