Test Harness Mutilation

Samuel Moelius

Mutation 2024 (May 28, 2024)

Necessist

https://github.com/trailofbits/publications/blob/master/reviews/2023-03-spool-platformv2-securityreview.pdf

Necessist

https://github.com/trailofbits/publications/blob/master/reviews/2023-03-spool-platformv2-securityreview.pdf

Necessist

Spool V2

Security Assessment

May 9, 2023

The `test_extendRewardEmission_ok` test does not check the new reward rate and duration to verify the effect of the call to the `extendRewardEmission` function on the `RewardManager` contract:

```
function test_extendRewardEmission_ok() public {
    deal(address(rewardToken), vaultOwner, rewardAmount * 2, true);
    vm.startPrank(vaultOwner);
    rewardToken.approve(address(rewardManager), rewardAmount * 2);
    rewardManager.addToken(smartVault, rewardToken, rewardDuration, rewardAmount);

    rewardManager.extendRewardEmission(smartVault, rewardToken, 1 ether,
rewardDuration);
    vm.stopPrank();
}
```

Figure 27.1: An insufficient test case for `extendRewardEmission`
spool-v2-core/RewardManager.t.sol

https://github.com/trailofbits/publications/blob/master/reviews/2023-03-spool-platformv2-securityreview.pdf

Necessist

Spool V2

Security Assessment

May 9, 2023

The `test_extendRewardEmission_ok` test does not check the new reward rate and duration to verify the effect of the call to the `extendRewardEmission` function on the `RewardManager` contract:

```
function test_extendRewardEmission_ok() public {
    deal(address(rewardToken), vaultOwner, rewardAmount * 2, true);
    vm.startPrank(vaultOwner);
    rewardToken.approve(address(rewardManager), rewardAmount * 2);
    rewardManager.addToken(smartVault, rewardToken, rewardDuration, rewardAmount);

    rewardManager.extendRewardEmission(smartVault, rewardToken, 1 ether,
rewardDuration);
    vm.stopPrank();
}
```

Recommendations
Short term, fix the test cases described above.

Long term, review all of the system's test cases and make sure that they verify the given state change correctly and sufficiently after an interaction with the protocol. Use Necessist to find broken test cases and fix them.

https://github.com/trailofbits/publications/blob/master/reviews/2023-03-spool-platformv2-securityreview.pdf

Outline

- Introduction and overview of Necessist
- Method
  - Parsing
  - Execution
- Limitations
- Future work and conclusion

Introduction

How do you prevent bugs in tests?

- Tests are software, software contains bugs, and so tests can contain bugs.
- We have lots of tools for finding bugs in conventional software.
- But how does one find bugs in tests?

Necessist overview

- Finding certain types of bugs in tests.
- Removes individual statements and method calls from a test and then sees whether the test passes.
- If such a **mutilated** test passes, it could contain a bug, e.g., because of an incorrect assumption held by the test's author.

  (Examples follow...)

Example (1 of 2): xNFTs

```
describe("the curator on an xNFT can be verified", () => {
  it("unless the signer does not match the curator...", async () => {
    try {
      await client.verify(xnft);
      assert.ok(false);
    } catch (_err) {}
  });
  ...
)
```

Example (1 of 2): xNFTs

```
describe("the curator on an xNFT can be verified", () => {
  it("unless the signer does not match the curator...", async () => {
    try {
      await client.verify(xnft);
      assert.ok(false);
    } catch (_err) {}
  });
  ...
)
```

This test checks that a call to `verify` fails if the caller is not the "curator."

Example (1 of 2): xNFTs

```
describe("the curator on an xNFT can be verified", () => {
  it("unless the signer does not match the curator...", async () => {
    try {
      await client.verify(xnft);
      assert.ok(false);
    } catch (_err) {}
  });
  ...
)
```

Example (1 of 2): xNFTs

```
describe("the curator on an xNFT can be verified", () => {
  it("unless the signer does not match the curator...", async () => {
    try {
      await client.verify(xnft);
      assert.ok(false);
    } catch (_err) {}
  });
  ...
)
```

If you remove this statement, the test still passes.

Example (1 of 2): xNFTs

```
describe("the curator on an xNFT can be verified", () => {
  it("unless the signer does not match the curator...", async () => {
    try {
      await client.verify(xnft);
      assert.ok(false);
    } catch (_err) {}
  });
  ...
)
```

Example (1 of 2): xNFTs

```
describe("the curator on an xNFT can be verified", () => {
  it("unless the signer does not match the curator...", async () => {
    try {
      await client.verify(xnft);
      assert.ok(false);
    } catch (err) {


    }
  });
  ...
)
```

Example (1 of 2): xNFTs

```
describe("the curator on an xNFT can be verified", () => {
  it("unless the signer does not match the curator...", async () => {
    try {
      await client.verify(xnft);
      assert.ok(false);
    } catch (err) {
      const e = err as anchor.AnchorError;
      assert.strictEqual(e.error.errorCode.code, "CuratorMismatch");
    }
  });
  ...
)
```

Checking the type of the exception fixes the problem.

Example (2 of 2): rust-openssl

```
#[test]
fn verify_trusted_callback_override_ok() {
    let server = Server::builder().build();

    let mut client = server.client();
    client.ctx().set_ca_file("test/root-ca.pem").unwrap();
    client
        .ctx()
        .set_verify_callback(SslVerifyMode::PEER, |_, x509| {
            assert!(x509.current_cert().is_some());
            true
        });

    client.connect();
}
```

Example (2 of 2): rust-openssl

```
#[test]
fn verify_trusted_callback_override_ok() {
    let server = Server::builder().build();

    let mut client = server.client();
    client.ctx().set_ca_file("test/root-ca.pem").unwrap();
    client
        .ctx()
        .set_verify_callback(SslVerifyMode::PEER, |_, x509| {
            assert!(x509.current_cert().is_some());
            true
        });

    client.connect();
}
```

This test checks that `set_verify_callback` is invoked with proper arguments.

Example (2 of 2): rust-openssl

```
#[test]
fn verify_trusted_callback_override_ok() {
    let server = Server::builder().build();

    let mut client = server.client();
    client.ctx().set_ca_file("test/root-ca.pem").unwrap();
    client
        .ctx()
        .set_verify_callback(SslVerifyMode::PEER, |_, x509| {
            assert!(x509.current_cert().is_some());
            true
        });

    client.connect();
}
```

Example (2 of 2): rust-openssl

```
#[test]
fn verify_trusted_callback_override_ok() {
    let server = Server::builder().build();

    let mut client = server.client();
    client.ctx().set_ca_file("test/root-ca.pem").unwrap();
    client
        .ctx()
        .set_verify_callback(SslVerifyMode::PEER, |_, x509| {
            assert!(x509.current_cert().is_some());
            true
        });

    client.connect();
}
```

If you remove this method call, the test still passes.

Example (2 of 2): rust-openssl

```
#[test]
fn verify_trusted_callback_override_ok() {
    let server = Server::builder().build();

    let mut client = server.client();
    client.ctx().set_ca_file("test/root-ca.pem").unwrap();
    client
        .ctx()
        .set_verify_callback(SslVerifyMode::PEER, |_, x509| {
            assert!(x509.current_cert().is_some());
            true
        });

    client.connect();
}
```

Example (2 of 2): rust-openssl

```
#[test]
fn verify_trusted_callback_override_ok() {

    let server = Server::builder().build();
    let mut client = server.client();
    client.ctx().set_ca_file("test/root-ca.pem").unwrap();
    client
        .ctx()
        .set_verify_callback(SslVerifyMode::PEER, |_, x509| {

            assert!(x509.current_cert().is_some());
            true
        });
    client.connect();

}
```

Example (2 of 2): rust-openssl

```
#[test]
fn verify_trusted_callback_override_ok() {
    static CALLED_BACK: AtomicBool = AtomicBool::new(false);
    let server = Server::builder().build();
    let mut client = server.client();
    client.ctx().set_ca_file("test/root-ca.pem").unwrap();
    client
        .ctx()
        .set_verify_callback(SslVerifyMode::PEER, |_, x509| {
            CALLED_BACK.store(true, Ordering::SeqCst);
            assert!(x509.current_cert().is_some());
            true
        });
    client.connect();
    assert!(CALLED_BACK.load(Ordering::SeqCst));
}
```

Adding a flag to verify the method was called fixes the problem.

Mutation testing

- Necessist implements a form of mutation testing.
- But Necessist differs from conventional mutation testing tools in a few key ways...

Comparison to conventional mutation testing (1 of 3)

- **Necessist tries to find bugs in tests, not improve test coverage.**
- Conventional mutation testing tools aim to improve a project's test coverage.
- In this sense, they target a project's test suite as whole, not individual tests.
- Necessist aims to find bugs in tests, and does target individual tests.

Comparison to conventional mutation testing (2 of 3)

- **Necessist performs exactly one type of mutation: removal.**
- Conventional mutation testing tools perform random modifications of a project's source code, e.g., changing `+` to `-` or `<` to `<=`.
- Necessist performs only removals, specifically of statements and method calls.

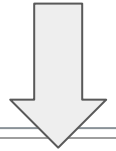
Comparison to conventional mutation testing (3 of 3)

- **Necessist mutates code deterministically.**
- Unlike conventional mutation testing tools, exhausting over all of Necessists mutants is **feasible**.
- For this reason, Necessist does not employ any sort of random selection.

Supported frameworks

Framework	Official Description	SUT Language	Testing Language
Anchor	A framework for Solana's Sealevel runtime providing several convenient developer tools for writing smart contracts	C/C++/Rust	TypeScript
Foundry	A blazing fast, portable and modular toolkit for Ethereum application development written in Rust	Solidity	Solidity
Go	An open source programming language that makes it easy to build simple, reliable, and efficient software	Go	Go
Hardhat	An Ethereum development environment for professionals	Solidity	TypeScript
Rust	A language empowering everyone to build reliable and efficient software	Rust	Rust

Supported frameworks



Framework	Official Description	SUT Language	Testing Language
Anchor	A framework for Solana's Sealevel runtime providing several convenient developer tools for writing smart contracts	C/C++/Rust	TypeScript
Foundry	A blazing fast, portable and modular toolkit for Ethereum application development written in Rust	Solidity	Solidity
Go	An open source programming language that makes it easy to build simple, reliable, and efficient software	Go	Go
Hardhat	An Ethereum development environment for professionals	Solidity	TypeScript
Rust	A language empowering everyone to build reliable and efficient software	Rust	Rust

Method

Method

- Necessist operates in two phases:
  - Parsing: identify removal candidates
  - Execution: run tests with candidates removed

Running example: net/http/transport_test.go

- Our paper describes experiments using 400 files from the Go standard library.
- We will use one of those files as a running example:
  - `net/http/transport_test.go` at commit `f736de0`
- This file has 5373 lines.

TestTransportServerClosingUnexpectedly

```
func TestTransportServerClosingUnexpectedly(t *testing.T) {
  setParallel(t)
  defer afterTest(t)
  ts := httptest.NewServer(hostPortHandler)
  defer ts.Close()
  c := ts.Client()

  fetch := func(n, retries int) string {
    condFatalf := func(format string, arg ...interface{}) {
      if retries <= 0 {
        t.Fatalf(format, arg...)
      }
      t.Logf("retrying shortly after expected error: "+format, arg...)
      time.Sleep(time.Second / time.Duration(retries))
    }
    for retries >= 0 {
      retries--
      res, err := c.Get(ts.URL)
      if err != nil {
        condFatalf("error in req #%d, GET: %v", n, err)
        continue
      }
      body, err := ioutil.ReadAll(res.Body)
      if err != nil {
        condFatalf("error in req #%d, ReadAll: %v", n, err)
        continue
      }
      res.Body.Close()
      return string(body)
    }
    panic("unreachable")
  }

  body1 := fetch(1, 0)
  body2 := fetch(2, 0)

  ts.CloseClientConnections() // surprise!

  // This test has an expected race. Sleeping for 25 ms prevents
  // it on most fast machines, causing the next fetch() call to
  // succeed quickly. But if we do get errors, fetch() will retry 5
  // times with some delays between.
  time.Sleep(25 * time.Millisecond)

  body3 := fetch(3, 5)

  if body1 != body2 {
    t.Errorf("expected body1 and body2 to be equal")
  }
  if body2 == body3 {
    t.Errorf("expected body2 and body3 to be different")
  }
}
```

Method: Parsing

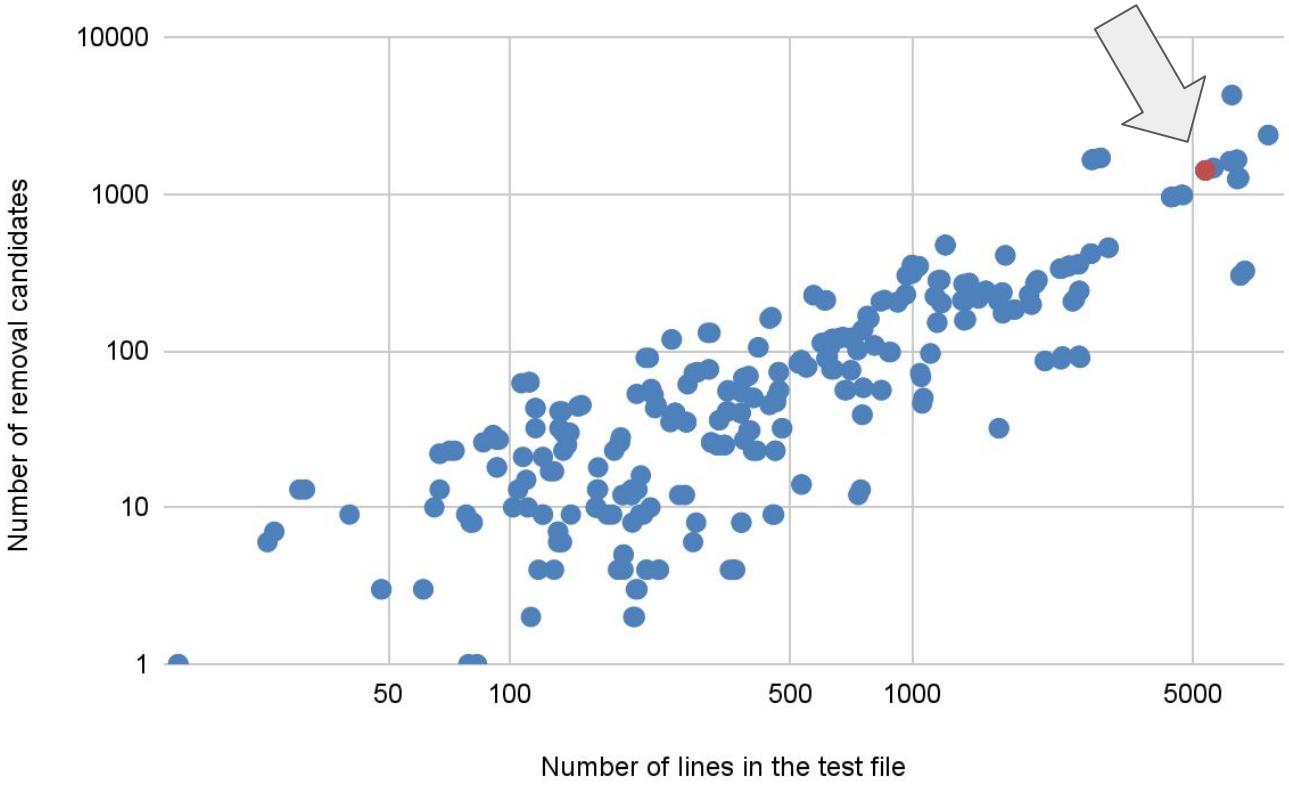
Parsing

- Generally speaking, Necessist will attempt to remove any statement **except** the following:
  - a statement containing other statements (e.g., a `for` loop)
  - a control statement (e.g., a `break`, `continue`, or `return`)
  - a declaration (e.g., a local or `let` binding)
  - an assertion
  - the last statement in a test
- Necessist will attempt to remove any method call, aside from certain framework-specific exceptions.

Running example

- Recall:
  - Our running example is `net/http/transport_test.go` at commit `f736de0`.
  - This file has 5373 lines.
- This file had 1414 removal candidates.

Removal candidates vs. lines



- The arrow points to our running example.
- The file with the most removal candidates was `cmd/go/go_test.go` at commit `746f405`.
- That file had 6252 lines and 4282 removal candidates.

TestTransportServerClosingUnexpectedly

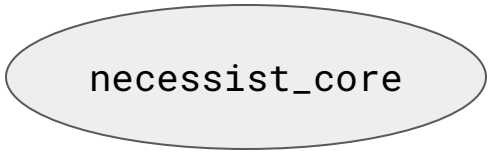
```
func TestTransportServerClosingUnexpectedly(t *testing.T) {
  setParallel(t)
  defer afterTest(t)
  ts := httptest.NewServer(hostPortHandler)
  defer ts.Close()
  c := ts.Client()

  fetch := func(n, retries int) string {
    condFatalf := func(format string, arg ...interface{}) {
      if retries <= 0 {
        t.Fatalf(format, arg...)
      }
      t.Logf("retrying shortly after expected error: "+format, arg...)
③     time.Sleep(time.Second / time.Duration(retries))
    }
    for retries >= 0 {
      retries--
      res, err := c.Get(ts.URL)
      if err != nil {
        condFatalf("error in req #%d, GET: %v", n, err)
        continue
      }
      body, err := ioutil.ReadAll(res.Body)
      if err != nil {
        condFatalf("error in req #%d, ReadAll: %v", n, err)
        continue
      }
      res.Body.Close()
      return string(body)
    }
    panic("unreachable")
  }

  body1 := fetch(1, 0)
  body2 := fetch(2, 0)

②ts.CloseClientConnections() // surprise!

  // This test has an expected race. Sleeping for 25 ms prevents
  // it on most fast machines, causing the next fetch() call to
  // succeed quickly. But if we do get errors, fetch() will retry 5
  // times with some delays between.
②time.Sleep(25 * time.Millisecond)

  body3 := fetch(3, 5)

  if body1 != body2 {
    t.Errorf("expected body1 and body2 to be equal")
  }
  if body2 == body3 {
    t.Errorf("expected body2 and body3 to be different")
  }
}
```

Parsing implementation

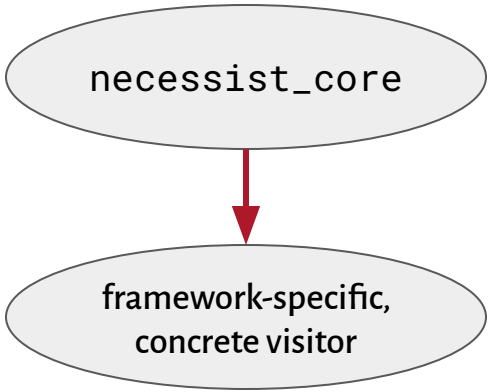
- Necessist implements the five “any statement except” rules in a language-agnostic manner.
- More specifically, the five rules are expressed in a component called the `GenericVisitor`.
- The `GenericVisitor` interacts with a frame-specific, “concrete” visitor through a set of opaque types.

(More details in the next few slides…)

Parsing implementation (continued)

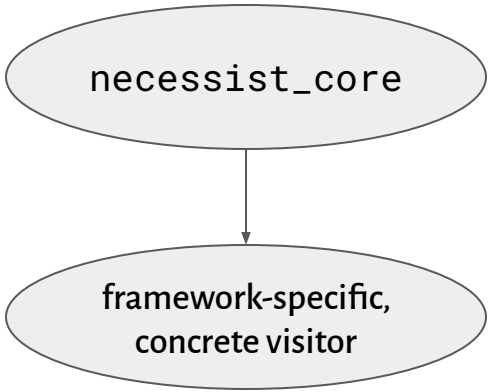


Parsing implementation (continued)



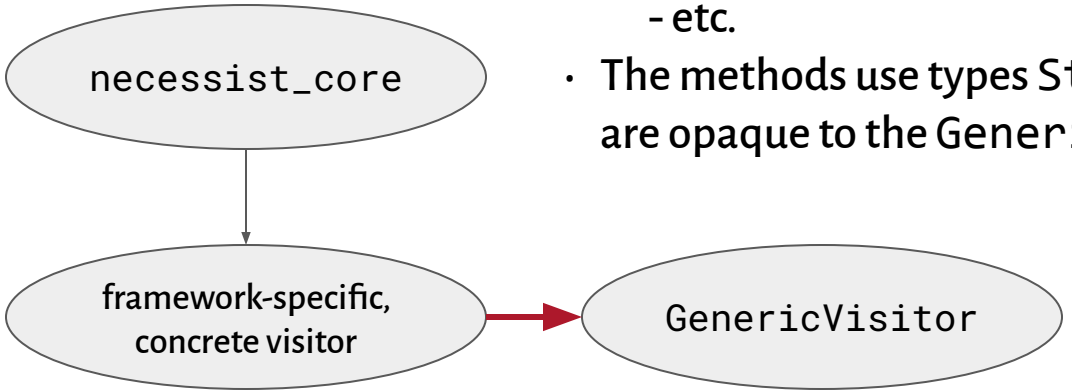
- `necessist_core` parses command line arguments and determines the framework that should be used.

Parsing implementation (continued)

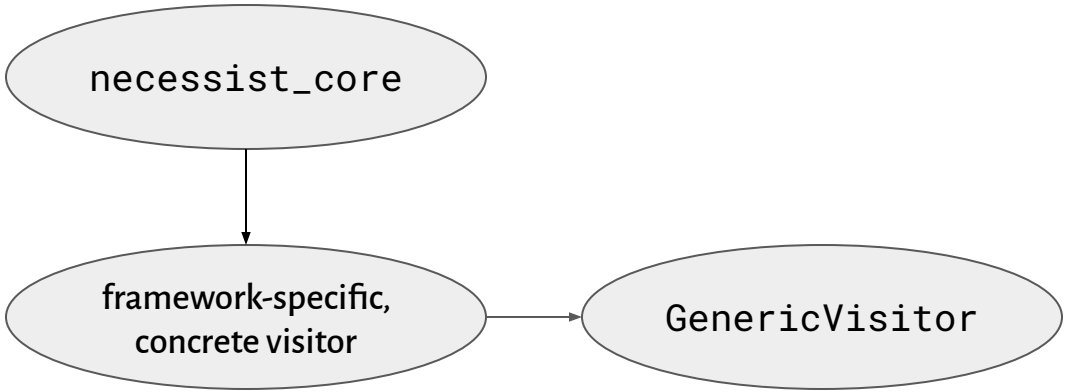


Parsing implementation (continued)

- The framework-specific visitor invokes GenericVisitor methods such as:
  - `visit_statement`/`visit_statement_post`
  - `visit_call`/`visit_call_post`
  - etc.
- The methods use types `Statement`, `Call`, etc., which are opaque to the `GenericVisitor`.



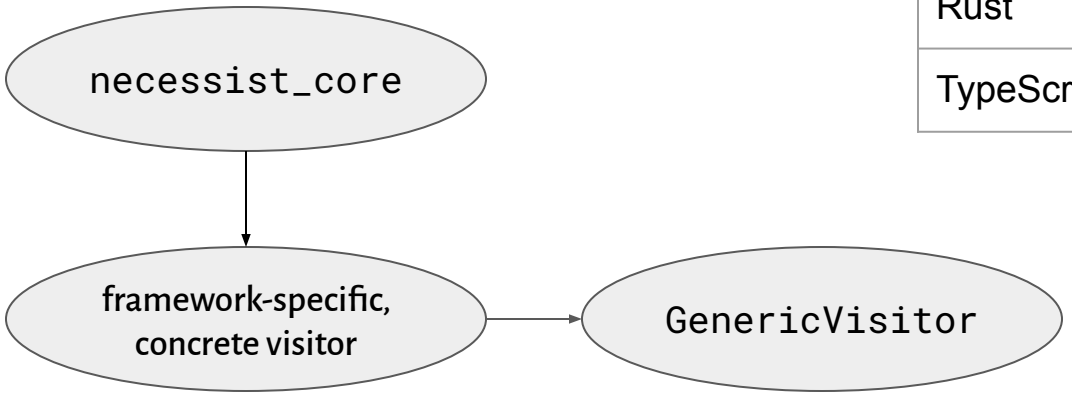
Parsing implementation (continued)



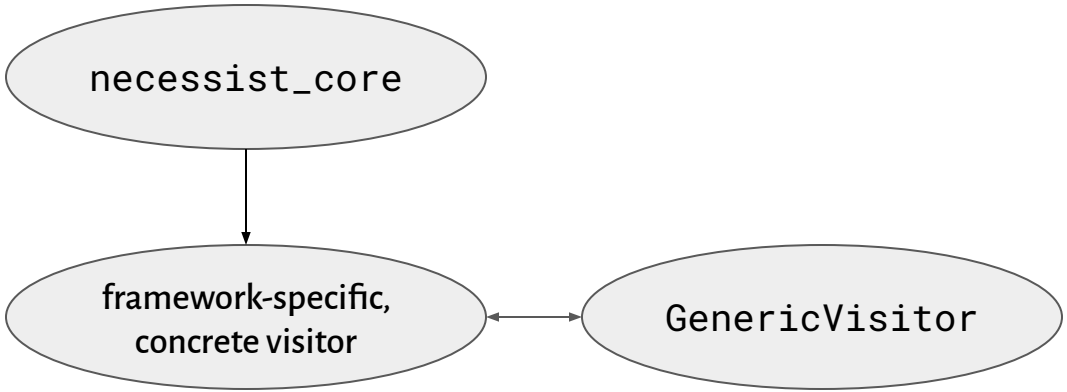
Parsing implementation (continued)

- The opaque types are based on the types that the concrete visitor's parser uses.
- For example, the Rust visitor's `Statement` type is the `syn` parsing library's `Stmt` type.

Testing Language	Frameworks	Parser
Go	Go	Tree-sitter
Solidity	Foundry	Solang
Rust	Rust	syn
TypeScript	Anchor, Hardhat	SWC

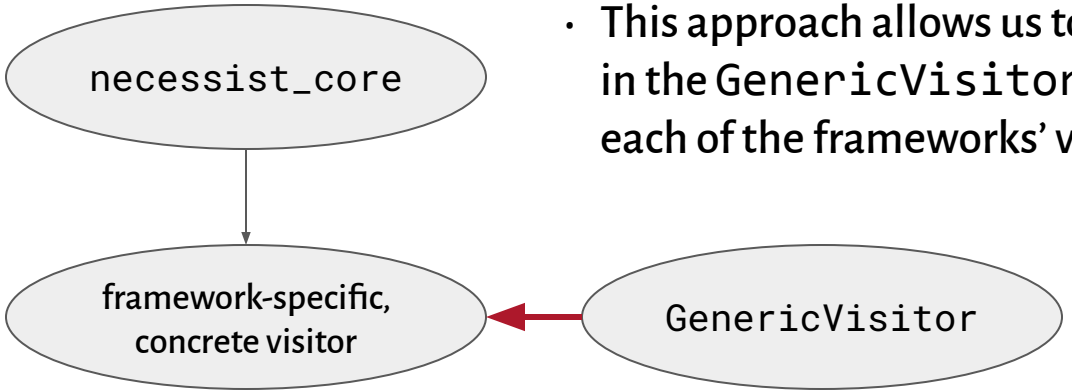


Parsing implementation (continued)



Parsing implementation (continued)

- The `GenericVisitor` can query the framework-specific visitor using functions as:
  - `statement_is_expression`
  - `expression_is_call`
  - etc.
- This approach allows us to express rules once in the `GenericVisitor`, as opposed to in each of the frameworks' visitors.



Method: Execution

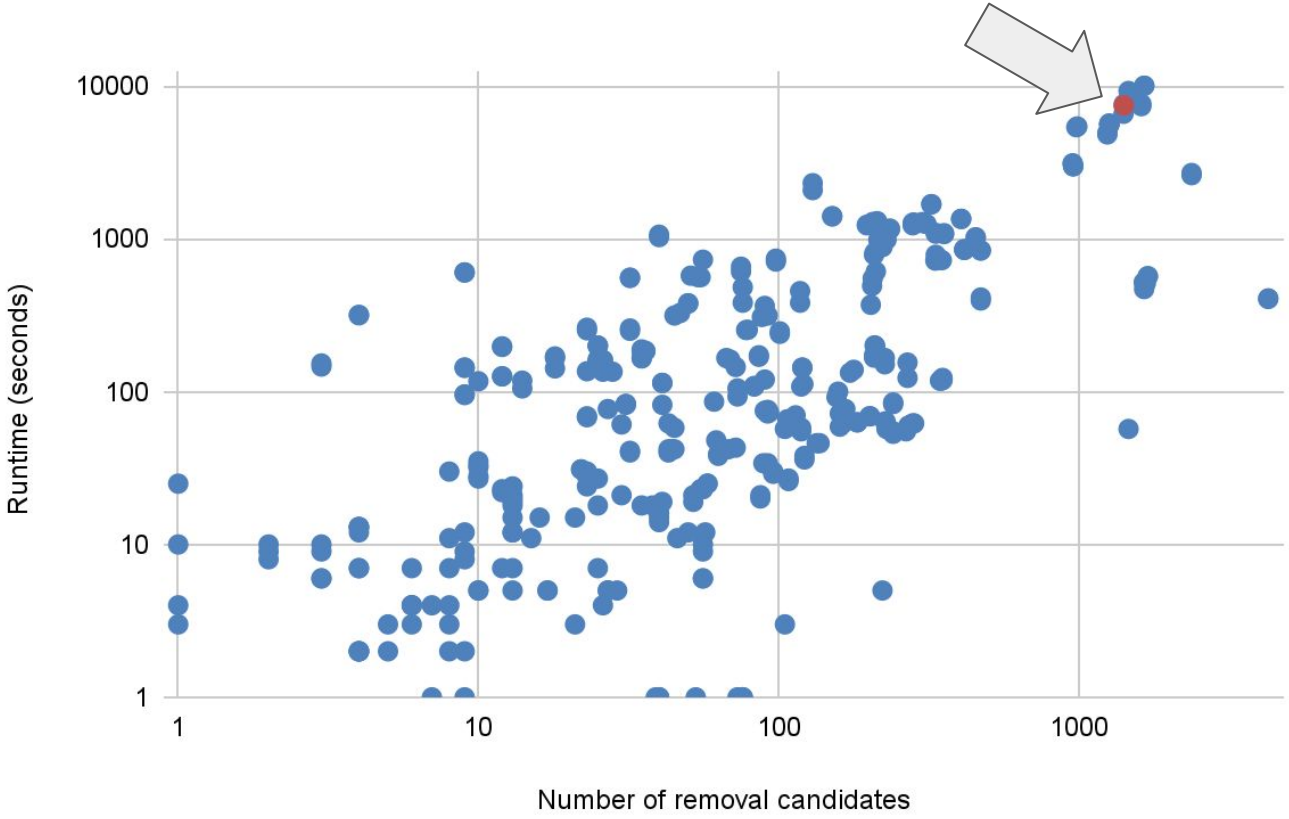
Execution

- For each removal candidate, there are four possible outcomes:
  - **NONBUILDABLE**: The test did not build.
  - **TIMEDOUT**: The test built, but timed out when run.
  - **FAILED**: The test built and ran to completion, but failed.
  - **PASSED**: The test built, ran to completion, and passed.
- Note that only **PASSED** outcomes can indicate bugs.

Running example

- Recall:
  - Our running example is `net/http/transport_test.go` at commit `f736de0`.
  - This file has 5373 lines.
  - This file had 1414 removal candidates.
- Necessist's runtime on this file was 7474 seconds (~2 hours).

Runtime vs. removal candidates



- The arrow points to our running example.
- The file with the longest runtime was at the same path, but at commit `d05c035`.
- That file had 1654 removal candidates and caused Necessist to run for 10039 seconds (~2.8 hours).

Running example

- Recall:
  - Our running example is `net/http/transport_test.go` at commit `f736de0`.
  - This file has 5373 lines.
  - This file had 1414 removal candidates.
  - Necessist's runtime on this file was 7474 seconds (~2 hours).
- The following were the removal candidates' outcomes:
  - **NONBUILDABLE**: 912
  - **TIMEDOUT**: 94
  - **FAILED**: 212
  - **PASSED**: 196

TestTransportServerClosing... NONBUILDABLE

```
func TestTransportServerClosingUnexpectedly(t *testing.T) {
  setParallel(t)
  defer afterTest(t)
  ts := httptest.NewServer(hostPortHandler)
  defer ts.Close()
  c := ts.Client()

  fetch := func(n, retries int) string {
    condFatalf := func(format string, arg ...interface{}) {
      if retries <= 0 {
        t.Fatalf(format, arg...)
      }
      t.Logf("retrying shortly after expected error: "+format, arg...)
②     time.Sleep(time.Second / time.Duration(retries))
    }
    for retries >= 0 {
      retries--
      res, err := c.Get(ts.URL)
      if err != nil {
        condFatalf("error in req #%d, GET: %v", n, err)
        continue
      }
      body, err := ioutil.ReadAll(res.Body)
      if err != nil {
        condFatalf("error in req #%d, ReadAll: %v", n, err)
        continue
      }
      res.Body.Close()
      return string(body)
    }
    panic("unreachable")
  }

  body1 := fetch(1, 0)
  body2 := fetch(2, 0)

  ts.CloseClientConnections() // surprise!

  // This test has an expected race. Sleeping for 25 ms prevents
  // it on most fast machines, causing the next fetch() call to
  // succeed quickly. But if we do get errors, fetch() will retry 5
  // times with some delays between.
  time.Sleep(25 * time.Millisecond)

  body3 := fetch(3, 5)

  if body1 != body2 {
    t.Errorf("expected body1 and body2 to be equal")
  }
  if body2 == body3 {
    t.Errorf("expected body2 and body3 to be different")
  }
}
```

TestTransportServerClosing... TIMEDOUT

```
func TestTransportServerClosingUnexpectedly(t *testing.T) {
  setParallel(t)
  defer afterTest(t)
  ts := httptest.NewServer(hostPortHandler)
  defer ts.Close()
  c := ts.Client()

  fetch := func(n, retries int) string {
    condFatalf := func(format string, arg ...interface{}) {
      if retries <= 0 {
        t.Fatalf(format, arg...)
      }
      t.Logf("retrying shortly after expected erro
      time.Sleep(time.Second / time.Duration(retri
    }
    for retries >= 0 {
      retries--
      res, err := c.Get(ts.URL)
      if err != nil {
        condFatalf("error in req #%d, GET: %v", n, err)
        continue
      }
      body, err := ioutil.ReadAll(res.Body)
      if err != nil {
        condFatalf("error in req #%d, ReadAll: %v", n, err)
        continue
      }
      res.Body.Close()
      return string(body)
    }
    panic("unreachable")
  }

  body1 := fetch(1, 0)
  body2 := fetch(2, 0)

  ctions() // surprise!

  expected race. Sleeping for 25 ms prevents
  machines, causing the next fetch() call to
  But if we do get errors, fetch() will retry 5
  delays between.
  e.Millisecond)

  body3 := fetch(3, 5)

  if body1 != body2 {
    t.Errorf("expected body1 and body2 to be equal")
  }
  if body2 == body3 {
    t.Errorf("expected body2 and body3 to be different")
  }
}
```

None of the removals resulted in a timeout.

TestTransportServerClosing... FAILED

```
func TestTransportServerClosingUnexpectedly(t *testing.T) {
  setParallel(t)
  defer afterTest(t)
  ts := httptest.NewServer(hostPortHandler)
  defer ts.Close()
  c := ts.Client()

  fetch := func(n, retries int) string {
    condFatalf := func(format string, arg ...interface{}) {
      if retries <= 0 {
        t.Fatalf(format, arg...)
      }
      t.Logf("retrying shortly after expected error: "+format, arg...)
      time.Sleep(time.Second / time.Duration(retries))
    }
    for retries >= 0 {
      retries--
      res, err := c.Get(ts.URL)
      if err != nil {
        condFatalf("error in req #%d, GET: %v", n, err)
        continue
      }
      body, err := ioutil.ReadAll(res.Body)
      if err != nil {
        condFatalf("error in req #%d, ReadAll: %v", n, err)
        continue
      }
      res.Body.Close()
      return string(body)
    }
    panic("unreachable")
  }

  body1 := fetch(1, 0)
  body2 := fetch(2, 0)

  ts.CloseClientConnections() // surprise!

  // This test has an expected race. Sleeping for 25 ms prevents
  // it on most fast machines, causing the next fetch() call to
  // succeed quickly. But if we do get errors, fetch() will retry 5
  // times with some delays between.
  time.Sleep(25 * time.Millisecond)

  body3 := fetch(3, 5)

  if body1 != body2 {
    t.Errorf("expected body1 and body2 to be equal")
  }
  if body2 == body3 {
    t.Errorf("expected body2 and body3 to be different")
  }
}
```

TestTransportServerClosing... PASSED

```
func TestTransportServerClosingUnexpectedly(t *testing.T) {
  setParallel(t)
  defer afterTest(t)
  ts := httptest.NewServer(hostPortHandler)
  defer ts.Close()
  c := ts.Client()

  fetch := func(n, retries int) string {
    condFatalf := func(format string, arg ...interface{}) {
      if retries <= 0 {
        t.Fatalf(format, arg...)
      }
      t.Logf("retrying shortly after expected error: "+format, arg...)
      time.Sleep(time.Second / time.Duration(retries))
    }
    for retries >= 0 {
      retries--
      res, err := c.Get(ts.URL)
      if err != nil {
        condFatalf("error in req #%d, GET: %v", n, err)
        continue
      }
      body, err := ioutil.ReadAll(res.Body)
      if err != nil {
        condFatalf("error in req #%d, ReadAll: %v", n, err)
        continue
      }
      res.Body.Close()
      return string(body)
    }
    panic("unreachable")
  }

  body1 := fetch(1, 0)
  body2 := fetch(2, 0)

  ts.CloseClientConnections() // surprise!

  // This test has an expected race. Sleeping for 25 ms prevents
  // it on most fast machines, causing the next fetch() call to
  // succeed quickly. But if we do get errors, fetch() will retry 5
  // times with some delays between.
  time.Sleep(25 * time.Millisecond)

  body3 := fetch(3, 5)

  if body1 != body2 {
    t.Errorf("expected body1 and body2 to be equal")
  }
  if body2 == body3 {
    t.Errorf("expected body2 and body3 to be different")
  }
}
```

Limitations

Limitations

- Based on our experience using Necessist, its main limitations are its:
  - Runtime
  - Tendency to produce false positives

Runtime

- Reasons for long runtimes include:
  - Recompilation
  - Timeouts caused by infinite loops caused by the removal of statements or method calls
  - Tests that are slow to start with, i.e., so repeatedly executing variants of them is slow

False positives

- A false positive is the removal of a statement or method call that results in a passing test, but that does not reflect a bug.

⚠️ WARNING ⚠️

The categorizations on the next several slides are subjective and empirical.

False positive (1 of 3): test setup

```
#[test]
fn test_sort() {
    let xs = (0..10).collect::<Vec<_>>();
    let mut ys = xs.clone();
    ys.shuffle();
    ys.sort();
    assert_eq!(xs, ys);
}
```

Necessist would report that removing this statement results in a passing test.

False positive (1 of 3): test setup

```
#[test]
fn test_sort() {
    let xs = (0..10).collect::<Vec<_>>();
    let mut ys = xs.clone();
    ys.shuffle();
    ys.sort();
    assert_eq!(xs, ys);
}
```

False positive (1 of 3): test setup

```rust
#[test]
fn test_sort() {
    let xs = (0..10).collect::<Vec<_>>();
    let mut ys = xs.clone();
    ys.shuffle();

    ys.sort();
    assert_eq!(xs, ys);
}
```

False positive (1 of 3): test setup

```
#[test]
fn test_sort() {
    let xs = (0..10).collect::<Vec<_>>();
    let mut ys = xs.clone();
    ys.shuffle();
    assert_ne!(xs, ys);
    ys.sort();
    assert_eq!(xs, ys);
}
```

Adding assertions fixes the problem.

False positive (2 of 3): subordinate checks

```rust
#[test]
fn test_allocator() {
    let mut allocator = Allocator::new();
    for _ in 0..N {
        perform_random_operation(&mut allocator);
        allocator.check_invariants();
    }
}
```

Necessist would report that removing this method call results in a passing test.

False positive (2 of 3): subordinate checks

```rust
#[test]
fn test_allocator() {
    let mut allocator = Allocator::new();
    for _ in 0..N {
        perform_random_operation(&mut allocator);
        allocator.check_invariants();
    }
}
```

False positive (2 of 3): subordinate checks

```rust
#[test]
fn test_allocator() {
    let mut allocator = Allocator::new();
    for _ in 0..N {
        perform_random_operation(&mut allocator);
        allocator.check_invariants();
    }
}
```

Necessist allows functions, macros, and methods to be ignored for this reason.

necessist.toml

```
ignored_methods = [“check_invariants“]
```

False positive (3 of 3): subordinate tests

```
#[test]
fn test_many() {
    for test_case in TEST_CASES {
        run_test(test_case);
    }
}
```

Necessist would report that removing this statement results in a passing test.

False positive (3 of 3): subordinate tests

```
#[test]
fn test_many() {
    for test_case in TEST_CASES {
        run_test(test_case);
    }
}
```

False positive (3 of 3): subordinate tests

```rust
#[test]
fn test_many() {
    for test_case in TEST_CASES {
        run_test(test_case);
    }
}
```

One could configure Necessist to ignore `run_test`…

necessist.toml

```
ignored_functions = ["run_test"]
```

False positive (3 of 3): subordinate tests

```
#[test]
fn test_many() {
    for test_case in TEST_CASES {
        run_test(test_case);
    }
}
```

One could configure Necessist to ignore `run_test`…

necessist.toml

```
ignored_functions = ["run_test"]
```

- But a better solution is planned…
- Often, functions that run that run subordinate tests are defined in the test files from which they are called.
- In such cases, Necessist could:
  - identify removal candidates in locally defined functions
  - treat the functions as though they were inlined at the points where they were called

Future work and conclusion

Future work

- Walk locally defined functions (as just described)
- Incorporate semantic information
  - Necessist currently operates only on syntax, and not, e.g., type or name resolution information.
- Explore other types of mutations
  - Observe that `x += 2;` is equivalent to `x += 1; x += 1;`.
  - So, should Necessist change `x += 2;` to `x += 1;`?
- How best to use Necessist in CI?
  - A common practice is to run in full on a schedule, or on a diff for each PR—are there other options?

Conclusion

- Necessist finds unnecessary statements and method calls in tests, which could indicate bugs in the tests.

- Repository: https://github.com/trailofbits/necessist
- My email: sam.moelius@trailofbits.com

Thank you for listening!