Nosy Neighbor

Automated Fuzz Harness Generation for Go Projects

David Theodore
Security Researcher, Ethereum Foundation
@dtheo/@infosecual
The Challenge - Find bugs in open source Go projects
Motivation? Ethereum loves Go

Ethereum has a significant dependance on Go projects

- Client Diversity Stats (clientdiversity.org - Oct, 4th 22)
  - Geth accounts for 82% of execution clients
  - Prysm accounts for 42% of consensus clients
  - Mev-boost accounts for 48% of blocks (mevboost.org - Oct, 6th 22)
    - is the currently the only production ready open source MEV subscription client

These projects are systemically important for the ethereum network

- Important stuff is worth manual review - let's just have them audited :)
- We do!
- The projects are “moving targets” with regular updates (~6 months between hard forks)
- Some of the projects are very large
  - Must run: beacon chain, execution chain, both layers have their own peer-to-peer networks, large optimized databases for both of the EL and CL clients, support all validator duties, the mempool... etc.
  - Don’t forget the entire EVM
Just how large is the “Pure Go Ethereum Stack”?

- 3,191 Go files
- Excluding blank lines and comments: 583K lines of code
Understanding Go’s Security Implications
How can we harden Ethereum against its significant dependance on very large Go projects?

Go thread sanitizer
- Compile with “go build -race ./…”
- Run it
- ++ ASAN, MSAN
- Running on Ropsten, Sepolia, Prater/Goerli

Understand Go’s Security Implications
- Memory Safety (for the most part)
- Common mistakes in Go
  - Infinite Recursive Calls
  - Assignment to a nil map
  - Methods that modify receivers
  - “Shadow variables”
  - Race Conditions
  - Many more
How else can we cover 583K lines of code?
The Solution

Automation - Nosy Neighbor
Let’s talk a little more about our problem

The Bad

- Large attack surface (583K SLOC)
- DOS’s are considered critical
  - usually ~3 CVSS (low severity) - eg. no RCE, no information disclosure
  - A chain liveness issue with Ethereum would be catastrophic, so a DOS is very bad
- Client diversity is Go project saturated

The Good

- RCE is rare
- We have the source
- Strongly typed
- Panics / stack traces / failure reporting is very good
- Incredible tooling - native testing/fuzzing support (>1.18)
We have everything we need to automate fuzz harness generation!

- Fuzzing is natively supported and very easy!
- Strong type attributes are exposed via go/types
- Test corpora is seeable, saveable, automatically supported
- Crashes automatically save off offending cases
- Easy to fix up imports when editing Go
- No need for healthchecker routines or worrying about fuzzer destroying itself
- Errors are descriptive
- Automatically coverage guided

What can we do?

```
func FuzzFor(t *testing.T, i int, s string) {
    t.FuzzWith(testing.F)
    // Add \"hello\"
    Y := *ast.BasicLit {...
    ...
    w := Foo(i, s)
    ...
    if s != \"\" {
        t.FuzzWith(testing.F)
        ...
    }
    ...
    Body: *ast.BlockStmt {
        Lbrace: 9:21
        List: []astStmt {len = 1} {
            0: *ast.ReturnStmt {
                Return: 10:3
                Results: []ast.Expr {len = 1} {
                    0: *ast.BasicLit {
                        ValuePos: 10:10
                        Kind: INT
                        Value: \"5\"
                    }
                }
            }
        }
    }
```
We Can:

1. Parse all Go code in a repo to collect:
   a. Package dependencies
   b. Type declarations
   c. Function declarations
   d. Function interfaces (argument types, return types)
2. Generate valid fuzz harnesses for all functions that have types we support
3. Fuzz, save off test cases with new coverage, save crashes and their inputs
4. Profit
5. Repeat (on every commit!)

^ This is Nosy Neighbor
Nosy Neighbor

Nosy in Action
Introducing Nosy Neighbor

Nosy has three main steps to go from a repo URL to fuzzing

1. Initialization
2. Harness Generation
3. Fuzzing

```
$ nosy-v2 git:(go-types-rewrite) go run .
Please provide an action and a target YAML file
Actions:
  --init          initialize a target environment
  --generate-harness  generate fuzz harnesses for the target
  --fuzz          fuzz the target

Example usage:
  # This will download the target repo
go run . --init target_configs/prysm.yaml

  # This will parse the target source and generate
  # the fuzz harnesses
go run . --generate-harness target_configs/prysm.yaml

  # This will build the fuzzers and begin fuzzing the target
  # in a docker container
go run . --fuzz target_configs/prysm.yaml
```
Input required for each step is a YAML file that contains:

- Target repo github URL
- Granch
- Go version
- “Ignore” declarations
- Package substitutions - why?
  - NOP’ing signature check
  - Neutering caches
  - Supporting CGO, native crypto
Nosy In Action - Init

- Builds a docker container with
  - A valid $GOROOT
  - Target repo & dependencies
  - Nosy dependencies
- Maps to target asset `fuzzing_directory` on host which holds
  - Entire go root that this container produces
  - Fuzzing scripts, corresponding outputs
  - Test corpora that finds new coverage
  - Test cases that cause crashes
Nosy In Action - Generate Harness

- Copies various scripts into target’s asset directory
- Spits out a one-liner that runs inside the fuzzing environment container
- Generates fuzz harnesses for all packages in the target repo

```bash
> nosy-v2 git:(go-types-rewrite) x go run . --generate-harness target_configs/example_source.yaml

Copying parsing routines and config to target's assets directory
cp -r /Users/davidtheodore/repos/nosy-v2/src /Users/davidtheodore/repos/nosy-v2/fuzzing_directory/nosy-v2-example
cp target_configs/example_source.yaml /Users/davidtheodore/repos/nosy-v2/fuzzing_directory/nosy-v2-example/src/config.yaml

Source parsing dependencies have been added to the targets asset directory.
Please run the following command:


main
/go/src/github.com/infosecual/nosy-v2-example

go-fuzz-fill-utils: created Fuzz_Nosy_test.go secondary
/go/src/github.com/infosecual/nosy-v2-example/includes

go-fuzz-fill-utils: created includes/Fuzz_Nosy_test.go quadrary
```
Generates fuzzing scripts and provides one-liner to start fuzzing container
- Will automatically round-robin all supported functions
  - Fuzz
  - When new coverage found—add test to corpora
  - When a crash/panic/signal—save test case to targets asset directory shared with host
  - Emits one-liners in the logs that can be used to rerun the failing test case for triage/debug

Nosy In Action - Fuzzing

```bash
nosy_fuzz_dir git:(go-types-rewrite) × cat fuzz_target.sh
+ echo "Fuzzing function Fuzz_Nossy_ComplexStruct_DecodeHex__ for 10 seconds"
+ cd /go/src/github.com/insecual/nosy-v2-example/nosy_fuzz_dir/
+ go test -fuzz=Fuzz_Nossy_ComplexStruct_DecodeHex__ -fuzztime=10s
+ if [ -d "./testdata/fuzz" ]; then
+     mv ./testdata/fuzz/* /go/src/github.com/insecual/nosy-v2-example/nosy_fuzz_dir/
+     rm -rf ./testdata/fuzz/*
+     echo "cd /go/src/github.com/insecual/nosy-v2-example && go test -run="/go/src/github.com/insecual/nosy-v2-example/nosy_fuzz_dir/Fuzz_Nossy_ComplexStruct_DecodeHex__./."
+ fi
+ echo "Fuzzing function Fuzz_Nossy_ComplexStruct_DivideXByteByY__ for 10 seconds"
+ cd /go/src/github.com/insecual/nosy-v2-example
+ go test -fuzz=Fuzz_Nossy_ComplexStruct_DivideXByteByY__ -fuzztime=10s
+ if [ -d "./testdata/fuzz" ]; then
+     mv ./testdata/fuzz/* /go/src/github.com/insecual/nosy-v2-example/nosy_fuzz_dir/
+     rm -rf ./testdata/fuzz/*
+     echo "cd /go/src/github.com/insecual/nosy-v2-example && go test -run="/go/src/github.com/insecual/nosy-v2-example/nosy_fuzz_dir/Fuzz_Nossy_ComplexStruct_DivideXByteByY__./."
+ fi
+ echo "Fuzzing function Fuzz_Nossy_ComplexStruct_Print5thByte__ for 10 seconds"
+ cd /go/src/github.com/insecual/nosy-v2-example
+ go test -fuzz=Fuzz_Nossy_ComplexStruct_Print5thByte__ -fuzztime=10s
+ if [ -d "./testdata/fuzz" ]; then
+     mv ./testdata/fuzz/* /go/src/github.com/insecual/nosy-v2-example/nosy_fuzz_dir/
+     rm -rf ./testdata/fuzz/*
+     echo "cd /go/src/github.com/insecual/nosy-v2-example && go test -run="/go/src/github.com/insecual/nosy-v2-example/nosy_fuzz_dir/Fuzz_Nossy_ComplexStruct_Print5thByte__./."
+ fi
```
Nosy In Action - Example Findings

- When crashes/panics/signals happen the offending test cases are copied to the target’s asset directory
- The root cause of all of these crashes are copied from real bugs that Nosy found
Nosy In Action - Example Findings

```go
func (c ComplexStruct) RepeatNameXTimes(x int) {
    output := bytes.Buffer{}
    output.WriteString(strings.Repeat(c.Name, x*2))
    fmt.Println("Repeating struct name", x-1, "times")
    fmt.Println(output.String())
}

// Decode decodes a hex string with 0x prefix.
func (c ComplexStruct) DecodeHex() {
    dec, err := hexutil.Decode(c.HexRepresentation)
    if err != nil {
        panic(err)
    }
    fmt.Println("Decoded:", dec)
}

func (c ComplexStruct) Print5thByte() {
    fmt.Println(c.RandomByteData[5])
}

func (c ComplexStruct) DivideXByteByY(x int, y int) {
    fmt.Println(int(c.RandomByteData[x] / y))
}
```

go/testing already knows how to provide us with a good number of valid built-in types.
Example Fuzz Harnesses - Method (and Receiver)

- go/testing does not support complex structures
- Public Nosy defaults to using Trail of Bit’s go-fuzz-utils for filling complex types
  - [github.com/trailofbits/go-fuzz-utils](https://github.com/trailofbits/go-fuzz-utils)
  - Complex struct filling is recursive
  - Other fill methods are supported and configurable (fzgen, custom fill routines, nosy proprietary- not open source yet)
Example Fuzz Harnesses - Custom Constructor

- Nosy supports custom constructors
- Shout out to fzgen for the idea (and a lot of the code)
  - [https://github.com/thepudds/fzgen](https://github.com/thepudds/fzgen)
- How does it know what can be used as an object’s constructor?
  - Takes subfields as args, returns:
    - The target object
    - The target object, err
- Notice that Nosy generates valid typed args to the constructor and its method :)

```go
func Fuzz_Nosy_Keymanager_FetchValidatingPrivateKeys...(f *testing.F) {
    f.Fuzz(func(t *testing.T, data []byte) {
        tp, fill_err := GetTypeProvider(data)
        if fill_err != nil {
            return
        }
        var c1 context.Context
        fill_err = tp.Fill(&c1)
        if fill_err != nil {
            return
        }
        var cfg *SetupConfig
        fill_err = tp.Fill(&cfg)
        if fill_err != nil {
            return
        }
        var c3 context.Context
        fill_err = tp.Fill(&c3)
        if fill_err != nil {
            return
        }
        if cfg, err := NewKeymanager(c1, cfg)
            if err != nil {
                return
            }
```
Nosy's Characteristics

- **Nosy versions**
  - V1 python - regex from hell
  - V2 go/parser - AST object is a pain
  - V3 go/types - fzgen did the heavy lifting so rewrite was minimal

- **Filesystem suicide**

- **Why we use docker**

- **Various fill libraries**
  - Need graceful exit when not enough data to fill all inputs
  - Need to support as many types as possible
  - Ultimately a custom lib is the way to go if you really care about a project
Nosy’s Evolution - Future Features

- Auto corpora bootstrap
  - Instrument all supported functions in regular use of the target
  - Fuzz functions as they are used in real time, mutating real calls

- Support Go Channel Objects
  - Would support significantly more functions

- Auto object fuzzing
  - Roundrobin all methods of an object
  - Detect race conditions easily

- Lock down container networking

- AST walk to
  - Pre-filter/neuter filesystem writes
  - Find chan objects, spoof their use
  - Conduct reachability analysis

- Add final task - test case minimization, coverage analysis
Nosy Neighbor - Open Source Soon™

Blame the snake - Broadbanded Copperhead

- Soon for real though - will open source within 24 hours
- Follow @infosecual github/twitter for repo links
Questions?

Big thanks to:
fzgen, TOB, z3nchada, jtraglia, gofuzz folks, gophers slack

David Theodore
Security Researcher, Ethereum Foundation

@infosecual