Nosy Neighbor

Automated Fuzz Harness Generation for Go Projects

David Theodore

Security Researcher, Ethereum Foundation

@dtheo/@infosecual

The Challenge

The Challenge - Find bugs in open source Go projects

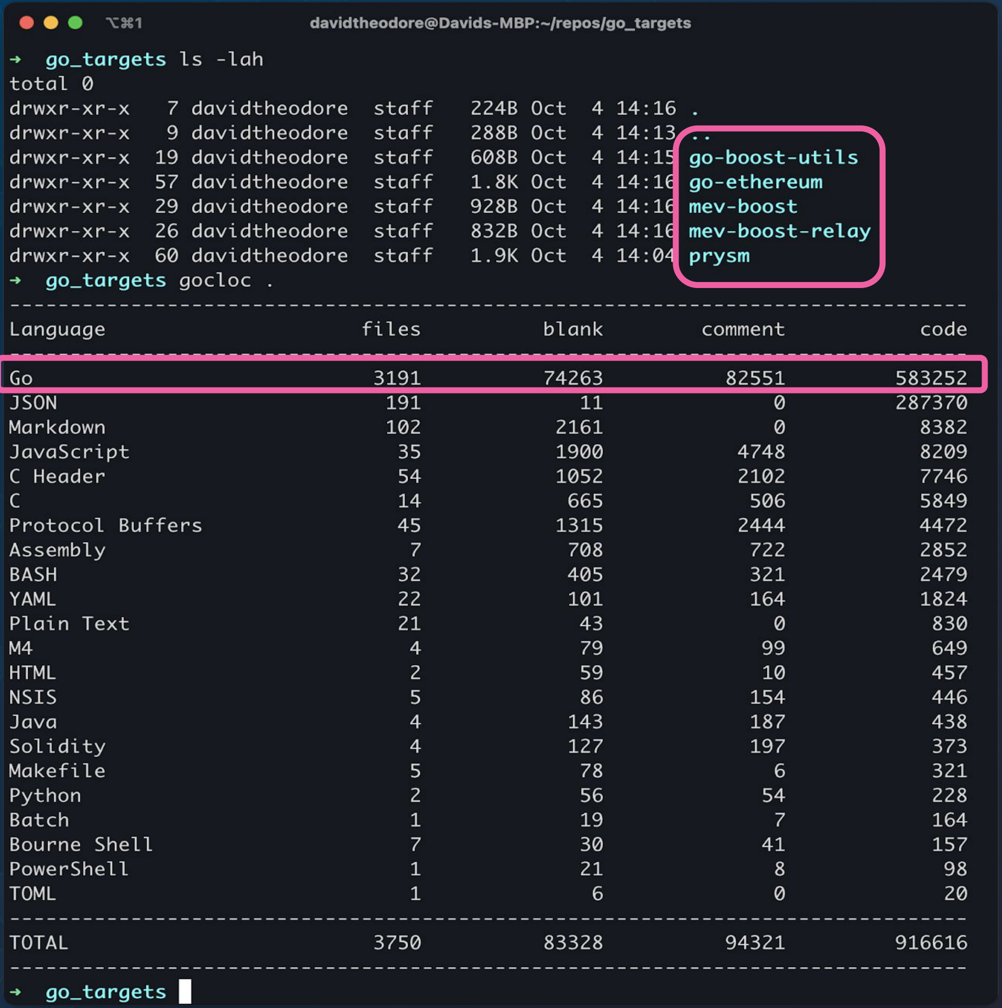
Motivation? Ethereum loves Go

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



```
davidtheodore@Davids-MBP:~/repos/go_targets
➜  go_targets ls -lah
total 0
drwxr-xr-x   7 davidtheodore  staff   224B Oct  4 14:16 .
drwxr-xr-x   9 davidtheodore  staff   288B Oct  4 14:13 ..
drwxr-xr-x  19 davidtheodore  staff   608B Oct  4 14:15 go-boost-utils
drwxr-xr-x  57 davidtheodore  staff   1.8K Oct  4 14:16 go-ethereum
drwxr-xr-x  29 davidtheodore  staff   928B Oct  4 14:16 mev-boost
drwxr-xr-x  26 davidtheodore  staff   832B Oct  4 14:16 mev-boost-relay
drwxr-xr-x  60 davidtheodore  staff   1.9K Oct  4 14:04 prysm
➜  go_targets gocloc .
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Go                            3191          74263          82551         583252
JSON                           191             11              0         287370
Markdown                       102           2161              0           8382
JavaScript                      35           1900           4748           8209
C Header                        54           1052           2102           7746
C                               14            665            506           5849
Protocol Buffers                45           1315           2444           4472
Assembly                         7            708            722           2852
BASH                            32            405            321           2479
YAML                            22            101            164           1824
Plain Text                      21             43              0            830
M4                               4             79             99            649
HTML                             2             59             10            457
NSIS                             5             86            154            446
Java                             4            143            187            438
Solidity                         4            127            197            373
Makefile                         5             78              6            321
Python                           2             56             54            228
Batch                            1             19              7            164
Bourne Shell                     7             30             41            157
PowerShell                       1             21              8             98
TOML                             1              6              0             20
-------------------------------------------------------------------------------
TOTAL                         3750          83328          94321         916616
-------------------------------------------------------------------------------
➜  go_targets
```

Just how large is the "Pure Go Ethereum Stack"?

- 3,191 Go files
- Excluding blank lines and comments:
  583K lines of code

The Investigation

Understanding Go’s Security Implications

How can we harden Ethereum against its significant dependance on very large Go projects?

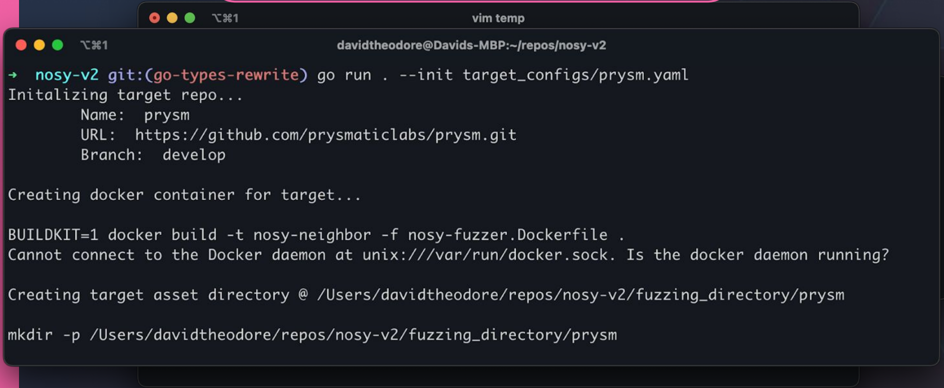
Understand Go's Security Implications

- 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

-Queryable
-Testable
-Nosy Neighbor

Go thread sanitizer

- Compile with "go build –race ./…"
- Run it (Nosy Neighbor) CodeQL, semgrep, gosec
- ++ ASAN, MSAN
- Running on Ropsten, Sepolia, Prater/Goerli



```
vim temp
davidtheodore@Davids-MBP:~/repos/nosy-v2
➜  nosy-v2 git:(go-types-rewrite) go run . --init target_configs/prysm.yaml
Initalizing target repo...
        Name:  prysm
        URL:  https://github.com/prysmaticlabs/prysm.git
        Branch:  develop

Creating docker container for target...

BUILDKIT=1 docker build -t nosy-neighbor -f nosy-fuzzer.Dockerfile .
Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?

Creating target asset directory @ /Users/davidtheodore/repos/nosy-v2/fuzzing_directory/prysm

mkdir -p /Users/davidtheodore/repos/nosy-v2/fuzzing_directory/prysm
```

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

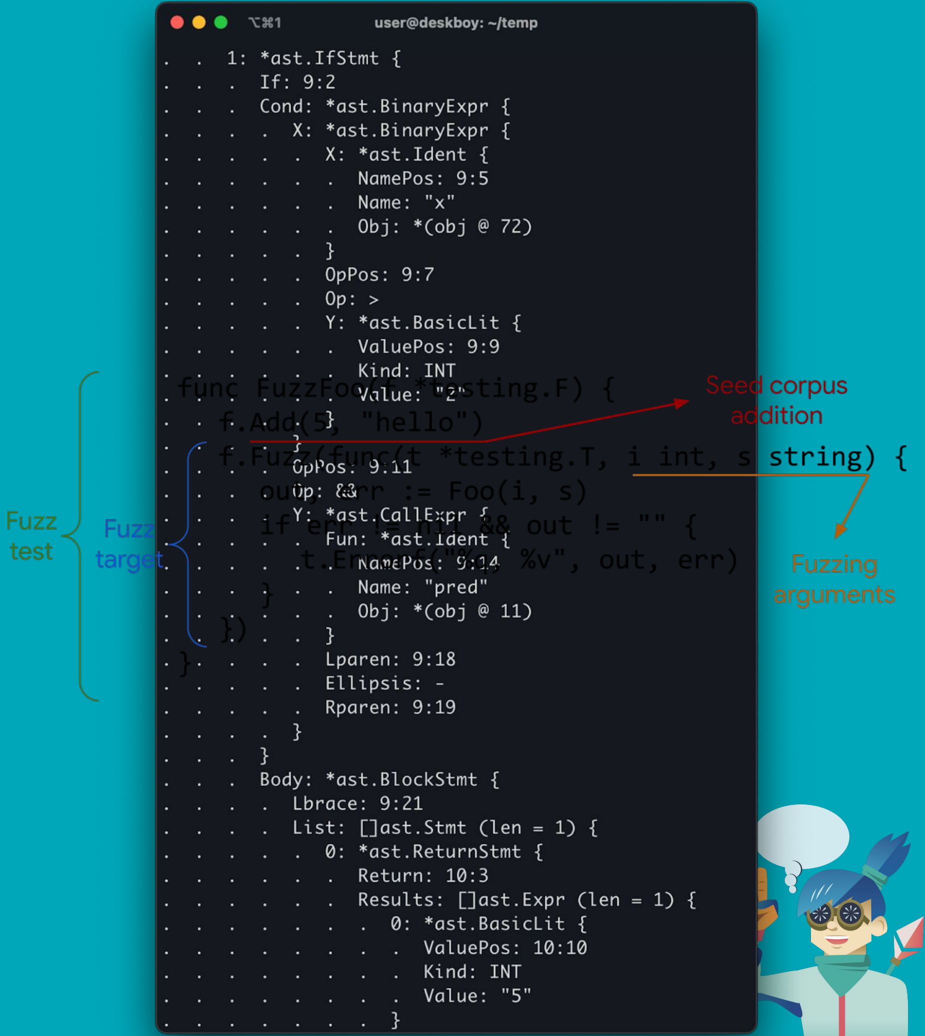
The Good

- 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!

- AST is exposed via go/parser, go/ast
- Strong type attributes are exposed via go/types
- Easy to fix up imports when editing Go
- What can we do?

- Fuzzing is natively supported and very easy!
  - Test Corpora is seeable, saveable, automatically supported
  - Crashes automatically save off offending cases
  - No need for healthchecker routines or worrying about fuzzer destroying itself
  - Errors are descriptive
  - Automatically coverage guided



```
.  .  1: *ast.IfStmt {
.  .  .  If: 9:2
.  .  .  Cond: *ast.BinaryExpr {
.  .  .  .  X: *ast.BinaryExpr {
.  .  .  .  .  X: *ast.Ident {
.  .  .  .  .  .  NamePos: 9:5
.  .  .  .  .  .  Name: "x"
.  .  .  .  .  .  Obj: *(obj @ 72)
.  .  .  .  .  }
.  .  .  .  .  OpPos: 9:7
.  .  .  .  .  Op: >
.  .  .  .  .  Y: *ast.BasicLit {
.  .  .  .  .  .  ValuePos: 9:9
.  .  .  .  .  .  Kind: INT
.  .  .  .  .  .  Value: "2"
.  .  .  .  .  }
.  .  .  .  }
.  .  .  .  OpPos: 9:11
.  .  .  .  Op: &&
.  .  .  .  Y: *ast.CallExpr {
.  .  .  .  .  Fun: *ast.Ident {
.  .  .  .  .  .  NamePos: 9:14
.  .  .  .  .  .  Name: "pred"
.  .  .  .  .  .  Obj: *(obj @ 11)
.  .  .  .  .  }
.  .  .  .  .  Lparen: 9:18
.  .  .  .  .  Ellipsis: -
.  .  .  .  .  Rparen: 9:19
.  .  .  .  }
.  .  .  }
.  .  .  Body: *ast.BlockStmt {
.  .  .  .  Lbrace: 9:21
.  .  .  .  List: []ast.Stmt (len = 1) {
.  .  .  .  .  0: *ast.ReturnStmt {
.  .  .  .  .  .  Return: 10:3
.  .  .  .  .  .  Results: []ast.Expr (len = 1) {
.  .  .  .  .  .  .  0: *ast.BasicLit {
.  .  .  .  .  .  .  .  ValuePos: 10:10
.  .  .  .  .  .  .  .  Kind: INT
.  .  .  .  .  .  .  .  Value: "5"
.  .  .  .  .  .  .  }
```

```go
func FuzzFoo(f *testing.F) {
    f.Add(5, "hello")
    f.Fuzz(func(t *testing.T, i int, s string) {
        out, err := Foo(i, s)
        if err != nil && out != "" {
            t.Errorf("%v %v", out, err)
        }
    })
}
```

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

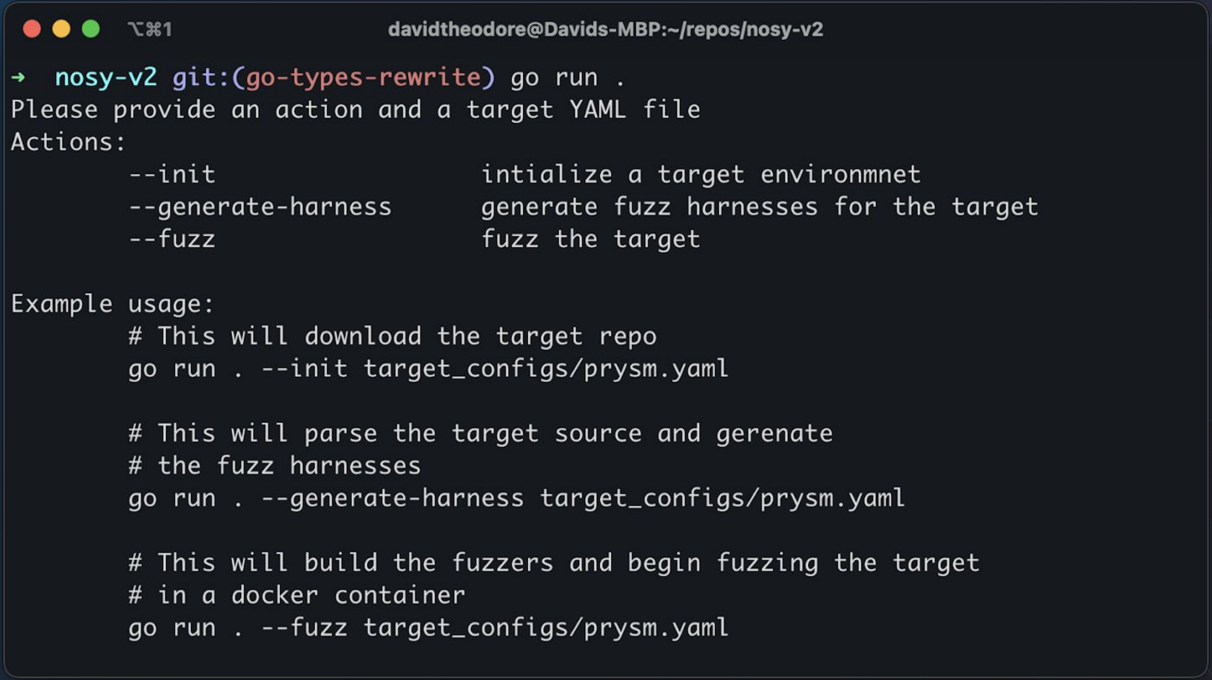
Nosy Neighbor

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

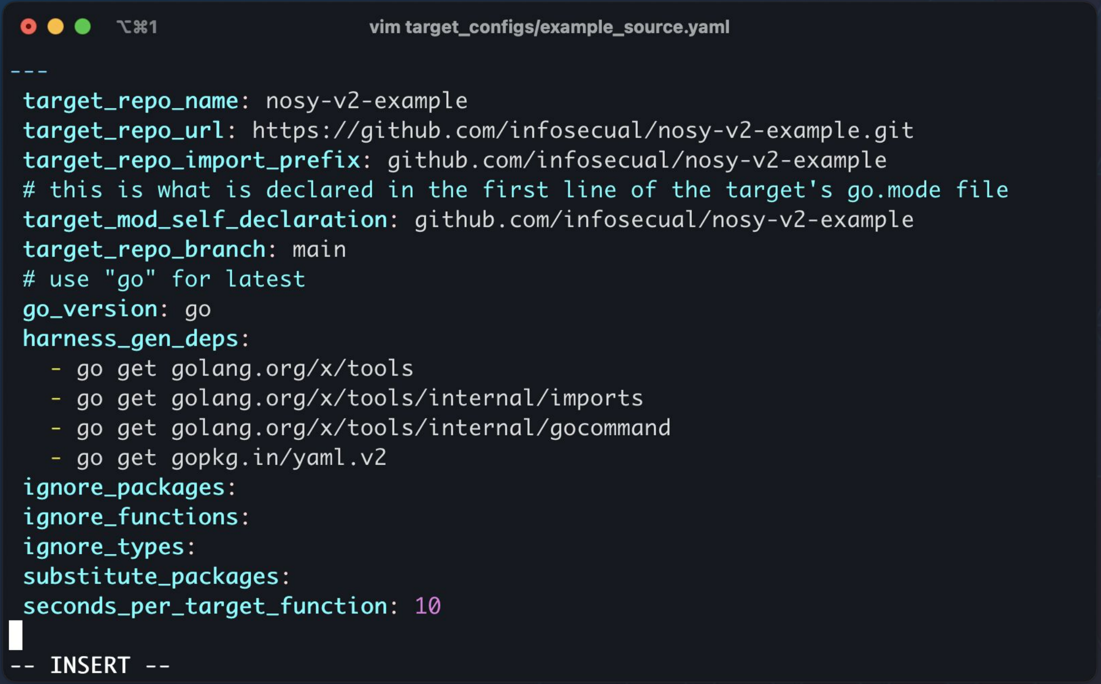


```
⌥⌘1        davidtheodore@Davids-MBP:~/repos/nosy-v2
→  nosy-v2 git:(go-types-rewrite) go run .
Please provide an action and a target YAML file
Actions:
        --init                  intialize a target environmnet
        --generate-harness      generate fuzz harnesses for the target
        --fuzz                  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 gerenate
        # 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
```

Nosy's Input: Target Config FIle

Input required for each step is a YAML file that contains:

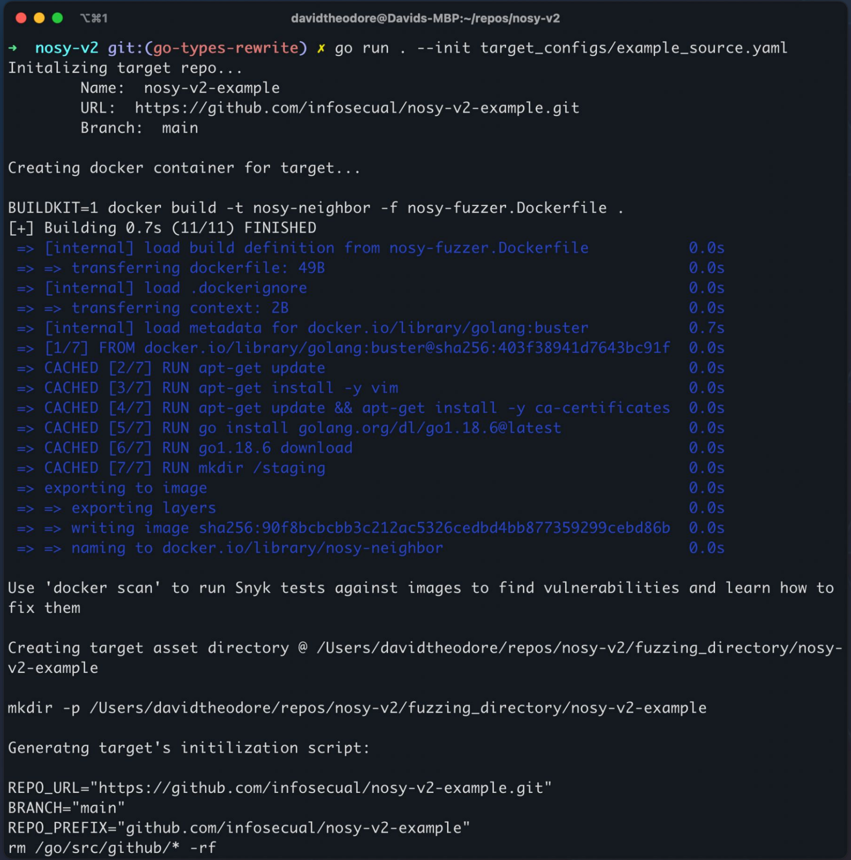
- Target repo github URL
- Granch
- Go version
- "Ignore" declarations
- Package substitutions - why?
  - NOP'ing signature check
  - Neutering caches
  - Supporting CGO, native crypto



```
---
target_repo_name: nosy-v2-example
target_repo_url: https://github.com/infosecual/nosy-v2-example.git
target_repo_import_prefix: github.com/infosecual/nosy-v2-example
# this is what is declared in the first line of the target's go.mode file
target_mod_self_declaration: github.com/infosecual/nosy-v2-example
target_repo_branch: main
# use "go" for latest
go_version: go
harness_gen_deps:
  - go get golang.org/x/tools
  - go get golang.org/x/tools/internal/imports
  - go get golang.org/x/tools/internal/gocommand
  - go get gopkg.in/yaml.v2
ignore_packages:
ignore_functions:
ignore_types:
substitute_packages:
seconds_per_target_function: 10

-- INSERT --
```

Nosy In Action - Init

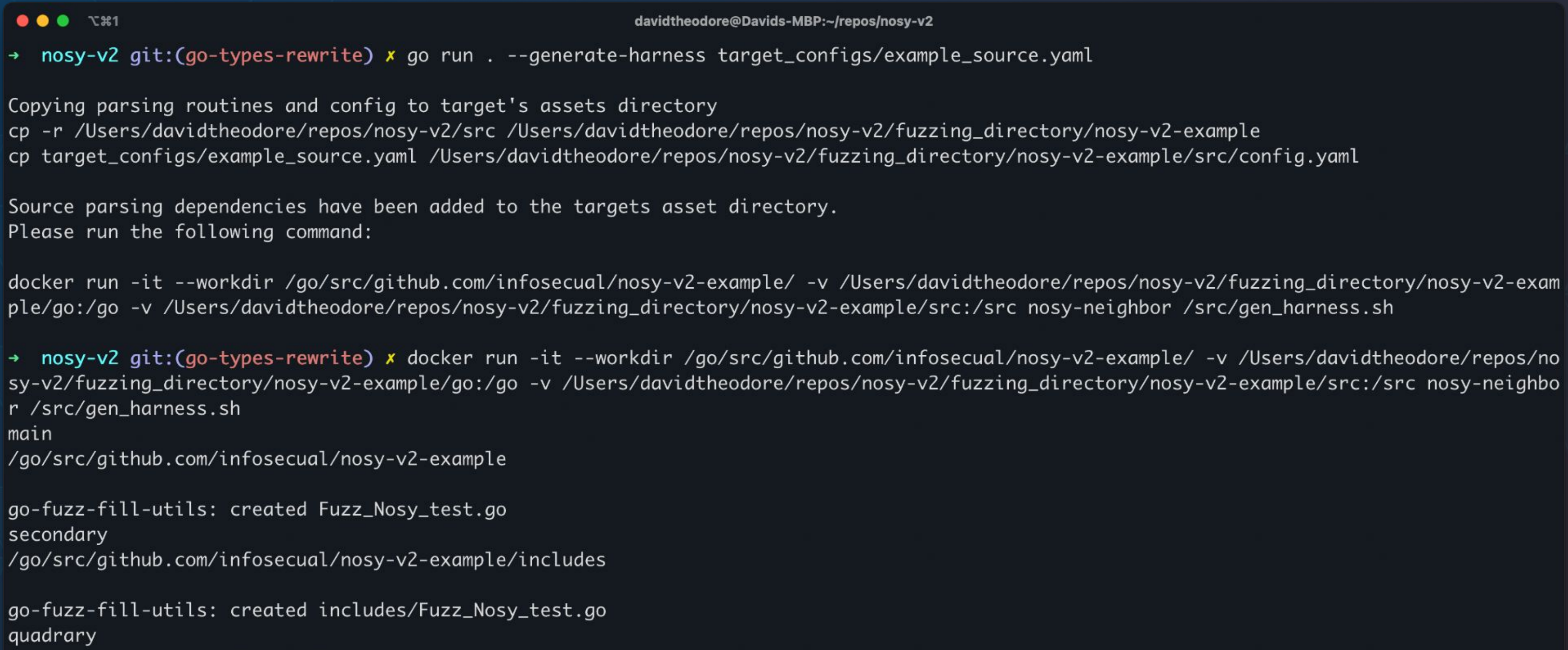
- Builds a docker container with
  - A valid $GOROOT
  - Target repo & dependencies
  - Nosy dependencies
- Maps to target asset *fuzzing_directory* on host which holds
  - Entire go root that this container produces
  - Fuzzing scripts, corresponding outputs
  - Test corpora that finds new coverage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



```
⌥⌘1        davidtheodore@Davids-MBP:~/repos/nosy-v2
➜  nosy-v2 git:(go-types-rewrite) ✗ go run . --init target_configs/example_source.yaml
Initalizing target repo...
        Name:  nosy-v2-example
        URL:  https://github.com/infosecual/nosy-v2-example.git
        Branch:  main

Creating docker container for target...

BUILDKIT=1 docker build -t nosy-neighbor -f nosy-fuzzer.Dockerfile .
[+] Building 0.7s (11/11) FINISHED
 => [internal] load build definition from nosy-fuzzer.Dockerfile          0.0s
 => => transferring dockerfile: 49B                                       0.0s
 => [internal] load .dockerignore                                         0.0s
 => => transferring context: 2B                                           0.0s
 => [internal] load metadata for docker.io/library/golang:buster          0.7s
 => [1/7] FROM docker.io/library/golang:buster@sha256:403f38941d7643bc91f 0.0s
 => CACHED [2/7] RUN apt-get update                                       0.0s
 => CACHED [3/7] RUN apt-get install -y vim                               0.0s
 => CACHED [4/7] RUN apt-get update && apt-get install -y ca-certificates 0.0s
 => CACHED [5/7] RUN go install golang.org/dl/go1.18.6@latest             0.0s
 => CACHED [6/7] RUN go1.18.6 download                                    0.0s
 => CACHED [7/7] RUN mkdir /staging                                       0.0s
 => exporting to image                                                    0.0s
 => => exporting layers                                                   0.0s
 => => writing image sha256:90f8bcbcbb3c212ac5326cedbd4bb877359299cebd86b 0.0s
 => => naming to docker.io/library/nosy-neighbor                          0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to
fix them

Creating target asset directory @ /Users/davidtheodore/repos/nosy-v2/fuzzing_directory/nosy-
v2-example

mkdir -p /Users/davidtheodore/repos/nosy-v2/fuzzing_directory/nosy-v2-example

Generatng target's initilization script:

REPO_URL="https://github.com/infosecual/nosy-v2-example.git"
BRANCH="main"
REPO_PREFIX="github.com/infosecual/nosy-v2-example"
rm /go/src/github/* -rf
```

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

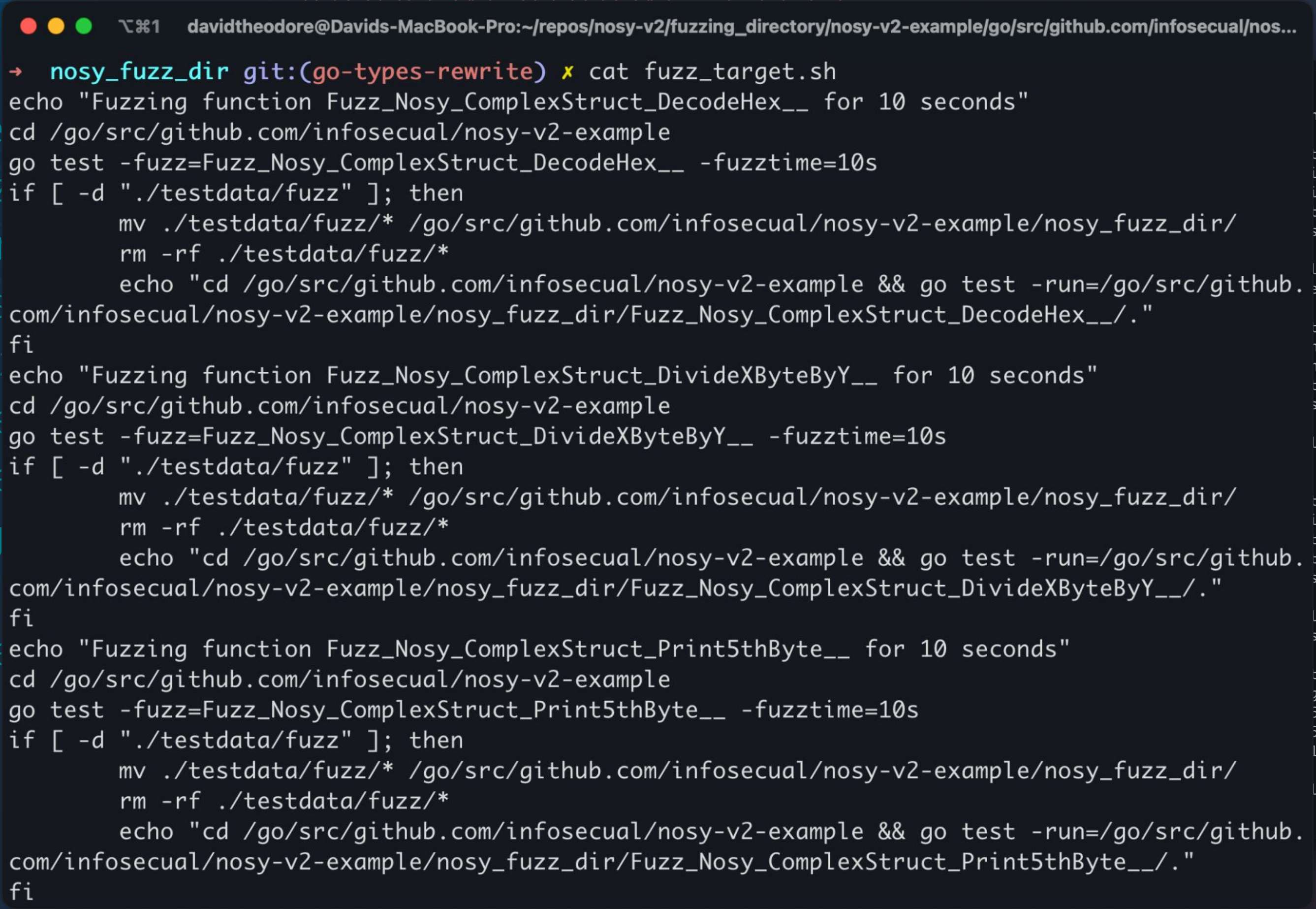


davidtheodore@Davids-MBP:~/repos/nosy-v2

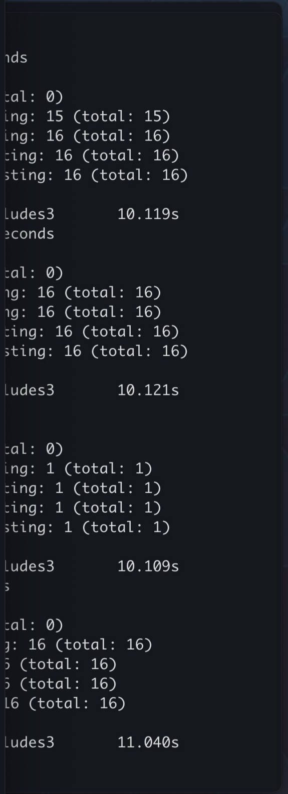
```
➜  nosy-v2 git:(go-types-rewrite) ✗ go run . --generate-harness target_configs/example_source.yaml

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:

docker run -it --workdir /go/src/github.com/infosecual/nosy-v2-example/ -v /Users/davidtheodore/repos/nosy-v2/fuzzing_directory/nosy-v2-exam
ple/go:/go -v /Users/davidtheodore/repos/nosy-v2/fuzzing_directory/nosy-v2-example/src:/src nosy-neighbor /src/gen_harness.sh

➜  nosy-v2 git:(go-types-rewrite) ✗ docker run -it --workdir /go/src/github.com/infosecual/nosy-v2-example/ -v /Users/davidtheodore/repos/no
sy-v2/fuzzing_directory/nosy-v2-example/go:/go -v /Users/davidtheodore/repos/nosy-v2/fuzzing_directory/nosy-v2-example/src:/src nosy-neighbo
r /src/gen_harness.sh
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
```

Nosy In Action – Fuzz

- Generate
  start fuzz
- Will autor
  functions
  - Fuz
  - Whe
  - Whe
    targ
  - Emi
    to re



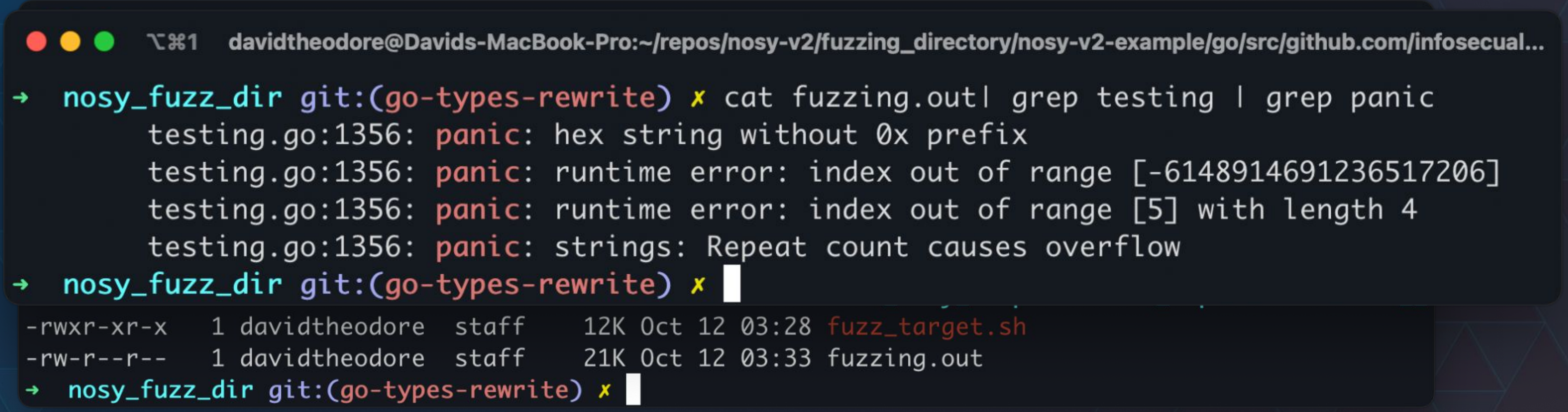
```
⌥⌘1   davidtheodore@Davids-MacBook-Pro:~/repos/nosy-v2/fuzzing_directory/nosy-v2-example/go/src/github.com/infosecual/nos...
➜  nosy_fuzz_dir git:(go-types-rewrite) ✗ cat fuzz_target.sh
echo "Fuzzing function Fuzz_Nosy_ComplexStruct_DecodeHex__ for 10 seconds"
cd /go/src/github.com/infosecual/nosy-v2-example
go test -fuzz=Fuzz_Nosy_ComplexStruct_DecodeHex__ -fuzztime=10s
if [ -d "./testdata/fuzz" ]; then
        mv ./testdata/fuzz/* /go/src/github.com/infosecual/nosy-v2-example/nosy_fuzz_dir/
        rm -rf ./testdata/fuzz/*
        echo "cd /go/src/github.com/infosecual/nosy-v2-example && go test -run=/go/src/github.
com/infosecual/nosy-v2-example/nosy_fuzz_dir/Fuzz_Nosy_ComplexStruct_DecodeHex__/."
fi
echo "Fuzzing function Fuzz_Nosy_ComplexStruct_DivideXByteByY__ for 10 seconds"
cd /go/src/github.com/infosecual/nosy-v2-example
go test -fuzz=Fuzz_Nosy_ComplexStruct_DivideXByteByY__ -fuzztime=10s
if [ -d "./testdata/fuzz" ]; then
        mv ./testdata/fuzz/* /go/src/github.com/infosecual/nosy-v2-example/nosy_fuzz_dir/
        rm -rf ./testdata/fuzz/*
        echo "cd /go/src/github.com/infosecual/nosy-v2-example && go test -run=/go/src/github.
com/infosecual/nosy-v2-example/nosy_fuzz_dir/Fuzz_Nosy_ComplexStruct_DivideXByteByY__/."
fi
echo "Fuzzing function Fuzz_Nosy_ComplexStruct_Print5thByte__ for 10 seconds"
cd /go/src/github.com/infosecual/nosy-v2-example
go test -fuzz=Fuzz_Nosy_ComplexStruct_Print5thByte__ -fuzztime=10s
if [ -d "./testdata/fuzz" ]; then
        mv ./testdata/fuzz/* /go/src/github.com/infosecual/nosy-v2-example/nosy_fuzz_dir/
        rm -rf ./testdata/fuzz/*
        echo "cd /go/src/github.com/infosecual/nosy-v2-example && go test -run=/go/src/github.
com/infosecual/nosy-v2-example/nosy_fuzz_dir/Fuzz_Nosy_ComplexStruct_Print5thByte__/."
fi
```



```
nds

al: 0)
ing: 15 (total: 15)
ing: 16 (total: 16)
ing: 16 (total: 16)
sting: 16 (total: 16)

ludes3      10.119s
econds

al: 0)
ng: 16 (total: 16)
ng: 16 (total: 16)
ing: 16 (total: 16)
sting: 16 (total: 16)

ludes3      10.121s

al: 0)
ing: 1 (total: 1)
ing: 1 (total: 1)
ing: 1 (total: 1)
sting: 1 (total: 1)

ludes3      10.109s
s

al: 0)
: 16 (total: 16)
5 (total: 16)
5 (total: 16)
16 (total: 16)

ludes3      11.040s
```

Nosy In Action - Example Findings

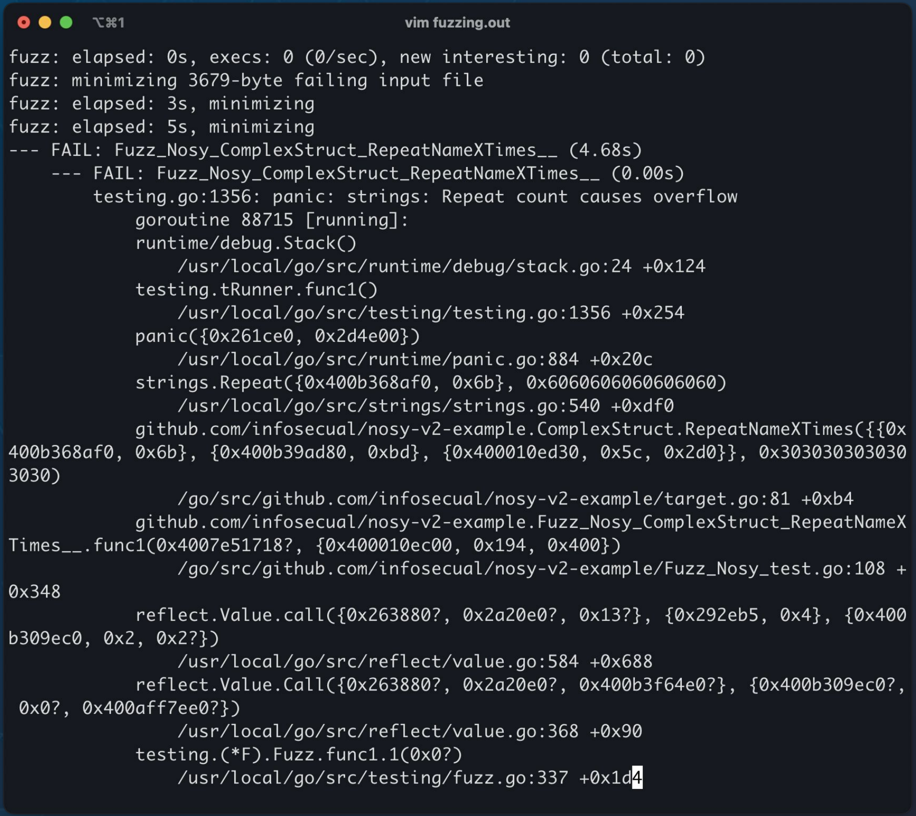
- When crashes/panics/signals happen the offending test cases are copied to the target's asset directory
- The root cause of all of these crashes are copied from real bugs that Nosy found



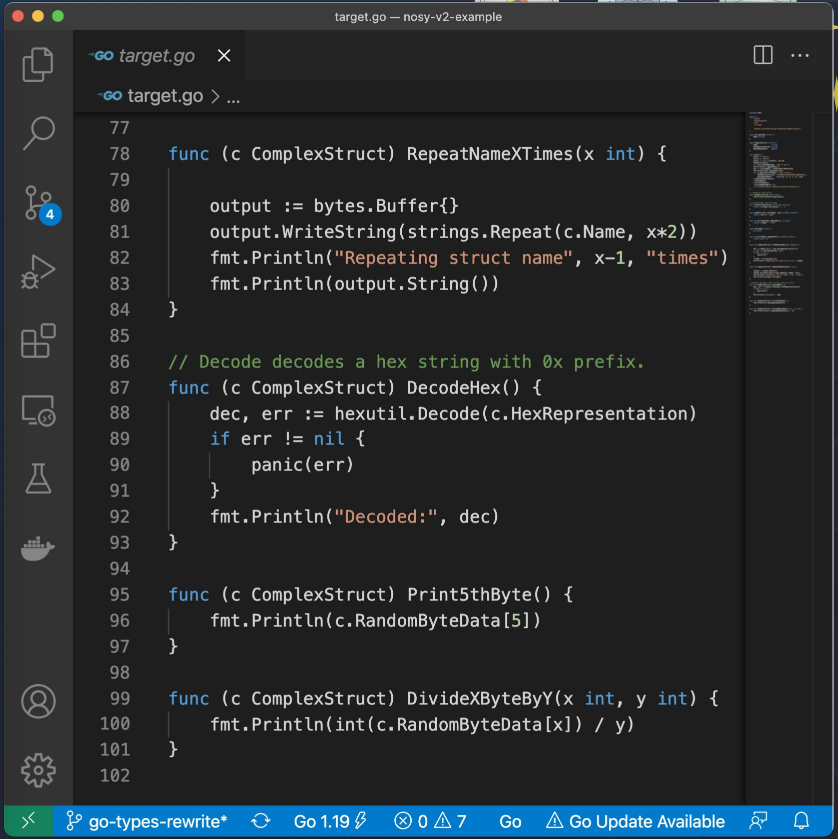
```
⌥⌘1   davidtheodore@Davids-MacBook-Pro:~/repos/nosy-v2/fuzzing_directory/nosy-v2-example/go/src/github.com/infosecual...
➜  nosy_fuzz_dir git:(go-types-rewrite) ✗ cat fuzzing.out| grep testing | grep panic
        testing.go:1356: panic: hex string without 0x prefix
        testing.go:1356: panic: runtime error: index out of range [-6148914691236517206]
        testing.go:1356: panic: runtime error: index out of range [5] with length 4
        testing.go:1356: panic: strings: Repeat count causes overflow
➜  nosy_fuzz_dir git:(go-types-rewrite) ✗
```

```
-rwxr-xr-x   1 davidtheodore  staff   12K Oct 12 03:28 fuzz_target.sh
-rw-r--r--   1 davidtheodore  staff   21K Oct 12 03:33 fuzzing.out
➜  nosy_fuzz_dir git:(go-types-rewrite) ✗
```

Nosy In Action - Example Findings



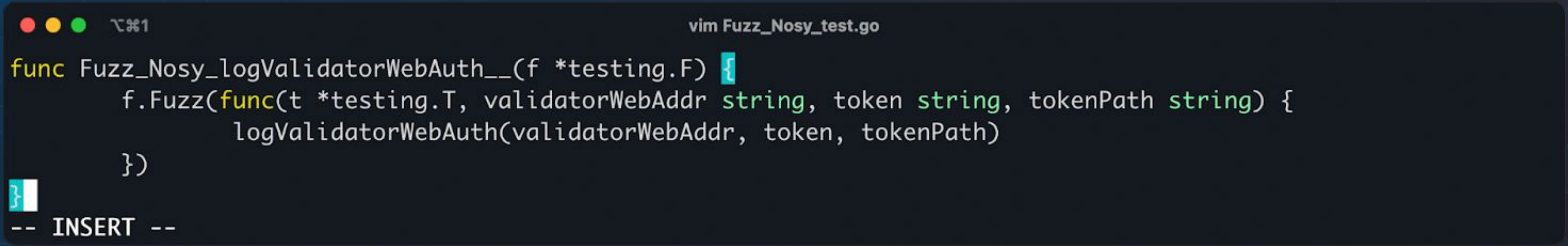
```
fuzz: elapsed: 0s, execs: 0 (0/sec), new interesting: 0 (total: 0)
fuzz: minimizing 3679-byte failing input file
fuzz: elapsed: 3s, minimizing
fuzz: elapsed: 5s, minimizing
--- FAIL: Fuzz_Nosy_ComplexStruct_RepeatNameXTimes__ (4.68s)
    --- FAIL: Fuzz_Nosy_ComplexStruct_RepeatNameXTimes__ (0.00s)
        testing.go:1356: panic: strings: Repeat count causes overflow
            goroutine 88715 [running]:
            runtime/debug.Stack()
                /usr/local/go/src/runtime/debug/stack.go:24 +0x124
            testing.tRunner.func1()
                /usr/local/go/src/testing/testing.go:1356 +0x254
            panic({0x261ce0, 0x2d4e00})
                /usr/local/go/src/runtime/panic.go:884 +0x20c
            strings.Repeat({0x400b368af0, 0x6b}, 0x6060606060606060)
                /usr/local/go/src/strings/strings.go:540 +0xdf0
            github.com/infosecual/nosy-v2-example.ComplexStruct.RepeatNameXTimes({{0x
400b368af0, 0x6b}, {0x400b39ad80, 0xbd}, {0x400010ed30, 0x5c, 0x2d0}}, 0x303030303030
3030)
                /go/src/github.com/infosecual/nosy-v2-example/target.go:81 +0xb4
            github.com/infosecual/nosy-v2-example.Fuzz_Nosy_ComplexStruct_RepeatNameX
Times__.func1(0x4007e51718?, {0x400010ec00, 0x194, 0x400})
                /go/src/github.com/infosecual/nosy-v2-example/Fuzz_Nosy_test.go:108 +
0x348
            reflect.Value.call({0x263880?, 0x2a20e0?, 0x13?}, {0x292eb5, 0x4}, {0x400
b309ec0, 0x2, 0x2?})
                /usr/local/go/src/reflect/value.go:584 +0x688
            reflect.Value.Call({0x263880?, 0x2a20e0?, 0x400b3f64e0?}, {0x400b309ec0?,
 0x0?, 0x400aff7ee0?})
                /usr/local/go/src/reflect/value.go:368 +0x90
            testing.(*F).Fuzz.func1.1(0x0?)
                /usr/local/go/src/testing/fuzz.go:337 +0x1d4
```



```go
77
78  func (c ComplexStruct) RepeatNameXTimes(x int) {
79
80      output := bytes.Buffer{}
81      output.WriteString(strings.Repeat(c.Name, x*2))
82      fmt.Println("Repeating struct name", x-1, "times")
83      fmt.Println(output.String())
84  }
85
86  // Decode decodes a hex string with 0x prefix.
87  func (c ComplexStruct) DecodeHex() {
88      dec, err := hexutil.Decode(c.HexRepresentation)
89      if err != nil {
90          panic(err)
91      }
92      fmt.Println("Decoded:", dec)
93  }
94
95  func (c ComplexStruct) Print5thByte() {
96      fmt.Println(c.RandomByteData[5])
97  }
98
99  func (c ComplexStruct) DivideXByteByY(x int, y int) {
100     fmt.Println(int(c.RandomByteData[x]) / y)
101 }
102
```

Example Fuzz Harnesses - Simple Function Function

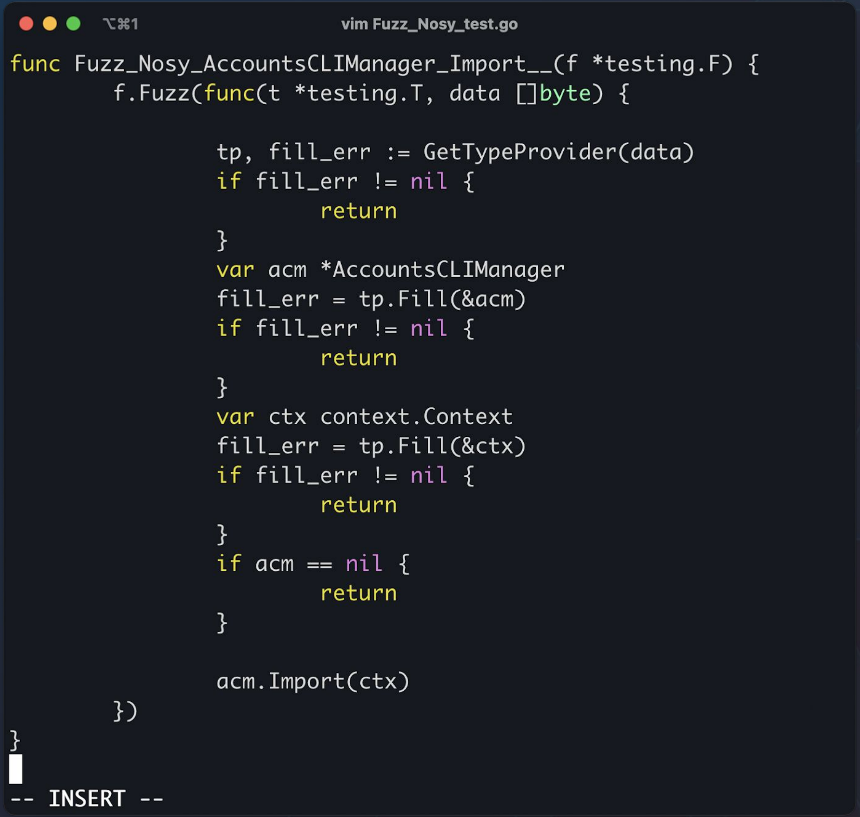
go/testing already knows how to provide us with a good number of valid built-in types



```
func Fuzz_Nosy_logValidatorWebAuth__(f *testing.F) {
	f.Fuzz(func(t *testing.T, validatorWebAddr string, token string, tokenPath string) {
		logValidatorWebAuth(validatorWebAddr, token, tokenPath)
	})
}
-- INSERT --
```

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
  - Complex struct filling is recursive
  - Other fill methods are supported and configurable (fzgen, custom fill routines, nosy proprietary- not open source yet)

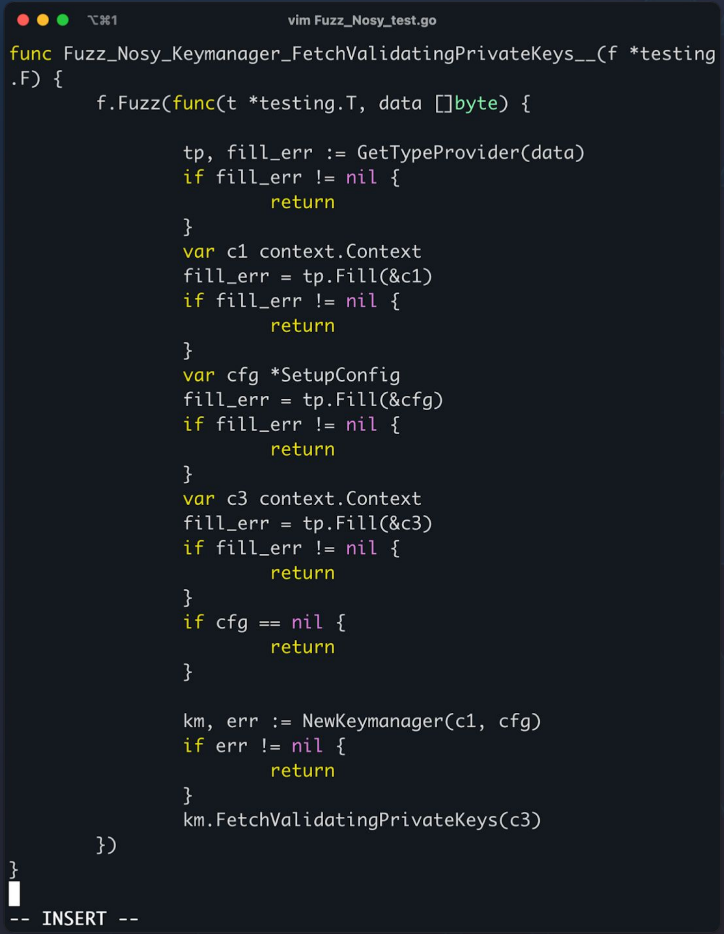


⌥⌘1 vim Fuzz_Nosy_test.go

```
func Fuzz_Nosy_AccountsCLIManager_Import__(f *testing.F) {
        f.Fuzz(func(t *testing.T, data []byte) {

                tp, fill_err := GetTypeProvider(data)
                if fill_err != nil {
                        return
                }
                var acm *AccountsCLIManager
                fill_err = tp.Fill(&acm)
                if fill_err != nil {
                        return
                }
                var ctx context.Context
                fill_err = tp.Fill(&ctx)
                if fill_err != nil {
                        return
                }
                if acm == nil {
                        return
                }

                acm.Import(ctx)
        })
}

-- INSERT --
```

Example Fuzz Harnesses - Custom Constructor

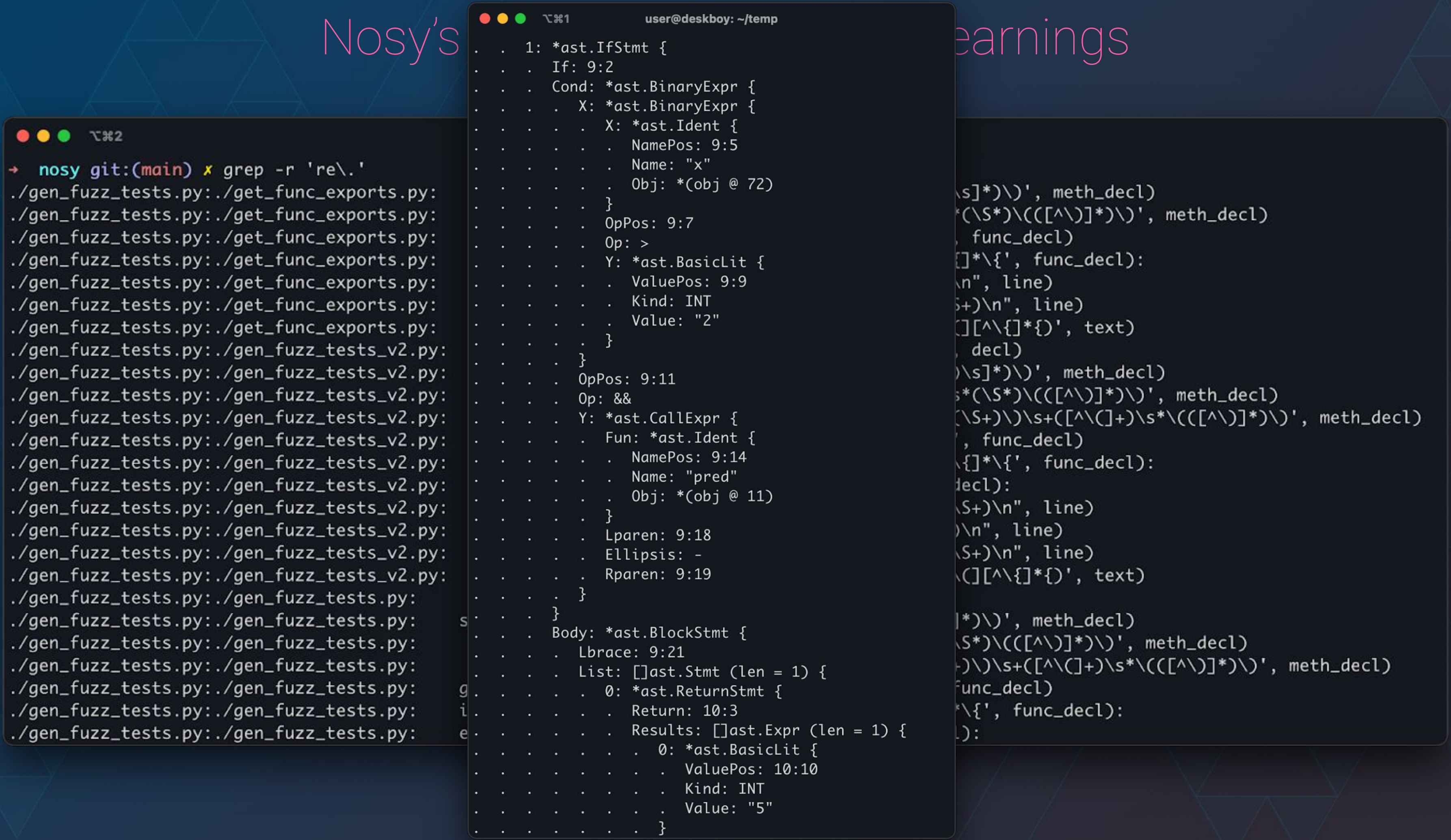
- Nosy supports custom constructors
- Shout out to fzgen for the idea (and a lot of the code)
  - https://github.com/thepudds/fzgen
- How does it know what can be used as an object's constructor?
  - Takes subfields as args, returns:
    - The target object
    - The target object, err
- Notice that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



vim Fuzz_Nosy_test.go

```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 == nil {
                        return
                }

                km, err := NewKeymanager(c1, cfg)
                if err != nil {
                        return
                }
                km.FetchValidatingPrivateKeys(c3)
        })
}

-- INSERT --
```

Nosy's earnings



```
⌥⌘2
→ nosy git:(main) ✗ grep -r 're\.'
./gen_fuzz_tests.py:./get_func_exports.py:
./gen_fuzz_tests.py:./get_func_exports.py:
./gen_fuzz_tests.py:./get_func_exports.py:
./gen_fuzz_tests.py:./get_func_exports.py:
./gen_fuzz_tests.py:./get_func_exports.py:
./gen_fuzz_tests.py:./get_func_exports.py:
./gen_fuzz_tests.py:./get_func_exports.py:
./gen_fuzz_tests.py:./gen_fuzz_tests_v2.py:
./gen_fuzz_tests.py:./gen_fuzz_tests_v2.py:
./gen_fuzz_tests.py:./gen_fuzz_tests_v2.py:
./gen_fuzz_tests.py:./gen_fuzz_tests_v2.py:
./gen_fuzz_tests.py:./gen_fuzz_tests_v2.py:
./gen_fuzz_tests.py:./gen_fuzz_tests_v2.py:
./gen_fuzz_tests.py:./gen_fuzz_tests_v2.py:
./gen_fuzz_tests.py:./gen_fuzz_tests_v2.py:
./gen_fuzz_tests.py:./gen_fuzz_tests_v2.py:
./gen_fuzz_tests.py:./gen_fuzz_tests_v2.py:
./gen_fuzz_tests.py:./gen_fuzz_tests_v2.py:
./gen_fuzz_tests.py:./gen_fuzz_tests.py:
./gen_fuzz_tests.py:./gen_fuzz_tests.py:    s
./gen_fuzz_tests.py:./gen_fuzz_tests.py:
./gen_fuzz_tests.py:./gen_fuzz_tests.py:
./gen_fuzz_tests.py:./gen_fuzz_tests.py:    g
./gen_fuzz_tests.py:./gen_fuzz_tests.py:    i
./gen_fuzz_tests.py:./gen_fuzz_tests.py:    e
```

```
⌥⌘1                user@deskboy: ~/temp
.  .  1: *ast.IfStmt {
.  .  .  If: 9:2
.  .  .  Cond: *ast.BinaryExpr {
.  .  .  .  X: *ast.BinaryExpr {
.  .  .  .  .  X: *ast.Ident {
.  .  .  .  .  .  NamePos: 9:5
.  .  .  .  .  .  Name: "x"
.  .  .  .  .  .  Obj: *(obj @ 72)
.  .  .  .  .  }
.  .  .  .  .  OpPos: 9:7
.  .  .  .  .  Op: >
.  .  .  .  .  Y: *ast.BasicLit {
.  .  .  .  .  .  ValuePos: 9:9
.  .  .  .  .  .  Kind: INT
.  .  .  .  .  .  Value: "2"
.  .  .  .  .  }
.  .  .  .  }
.  .  .  .  OpPos: 9:11
.  .  .  .  Op: &&
.  .  .  .  Y: *ast.CallExpr {
.  .  .  .  .  Fun: *ast.Ident {
.  .  .  .  .  .  NamePos: 9:14
.  .  .  .  .  .  Name: "pred"
.  .  .  .  .  .  Obj: *(obj @ 11)
.  .  .  .  .  }
.  .  .  .  .  Lparen: 9:18
.  .  .  .  .  Ellipsis: -
.  .  .  .  .  Rparen: 9:19
.  .  .  .  }
.  .  .  }
.  .  .  Body: *ast.BlockStmt {
.  .  .  .  Lbrace: 9:21
.  .  .  .  List: []ast.Stmt (len = 1) {
.  .  .  .  .  0: *ast.ReturnStmt {
.  .  .  .  .  .  Return: 10:3
.  .  .  .  .  .  Results: []ast.Expr (len = 1) {
.  .  .  .  .  .  .  0: *ast.BasicLit {
.  .  .  .  .  .  .  .  ValuePos: 10:10
.  .  .  .  .  .  .  .  Kind: INT
.  .  .  .  .  .  .  .  Value: "5"
.  .  .  .  .  .  .  }
```

```
\s]*)\)', meth_decl)
*(\S*)\(([^\)]*)\)', meth_decl)
, func_decl)
[]*\{', func_decl):
\n", line)
S+)\n", line)
(][^\{]*{)', text)
, decl)
)\s]*)\)', meth_decl)
s*(\S*)\(([^\)]*)\)', meth_decl)
(\S+)\)\s+([^\(]+)\s*\(([^\)]*)\)', meth_decl)
', func_decl)
\{]*\{', func_decl):
decl):
\S+)\n", line)
)\n", line)
\S+)\n", line)
\(][^\{]*{)', text)

]*)\)', meth_decl)
\S*)\(([^\)]*)\)', meth_decl)
+)\)\s+([^\(]+)\s*\(([^\)]*)\)', meth_decl)
func_decl)
*\{', func_decl):
):
```

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