Coverage Guided Fuzzing with GoFuzz

NoVA Hackers
14 May 2018

Parsia

Outline

- Intro
- What is Go-Fuzz?
- Fuzzing iprange
- Demo
- More demos if time permits
- Questions

Intro

What is Fuzzing?

- Throwing random input at software
- Basically 70% of security
- Hopefully intelligent

What is Go-Fuzz?

- AFL for Go
- Coverage Guided Fuzzing
- https://github.com/dvyukov/go-fuzz

What does Go-Fuzz do?

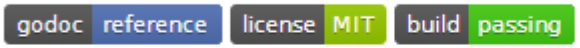
- Reports panics (Go crashes)
- Receives feedback on fuzzing input
- Adds good input to corpus

Fuzzing iprange

iprange package

- https://github.com/malfunkt/iprange

iprange



`iprange` is a library you can use to parse IPv4 addresses from a string in the `nmap` format.

It takes a string, and returns a list of `Min` - `Max` pairs, which can then be expanded and normalized automatically by the package.

Supported Formats

`iprange` supports the following formats:

- `10.0.0.1`
- `10.0.0.0/24`
- `10.0.0.*`
- `10.0.0.1-10`
- `10.0.0.1, 10.0.0.5-10, 192.168.1.*, 192.168.10.0/24`

iprange usage

```
package main

import (
    "log"

    "github.com/malfunkt/iprange"
)

func main() {
    list, err := iprange.ParseList("10.0.0.5-10, 192.168.10.0/24")
    if err != nil {
        log.Printf("error: %s", err)
    }
    log.Printf("%+v", list)

    rng := list.Expand()
    log.Printf("%s", rng)
}
```

Run

iprange Fuzz.go

Return value is feedback to fuzzer:

- 1 : Good input, add to corpus
- 0 : Bad input, ignore unless it gave us more coverage
- -1: Bad input, ignore even if it gave us more coverage

```
// +build gofuzz

package iprange

func Fuzz(data []byte) int {
    _, err := ParseList(string(data))
    if err != nil {
        return 0
    }
    return 1
}
```

Advantages of Go-Fuzz

- Fuzz internal packages because Fuzz is part of the package
- Fuzz specific functions (e.g. packet parser instead of fuzzing the network socket)
- Perform pre-processing on data before passing them to package (e.g. checksums, field lengths)
- Easy to setup
- Feedback on good/bad input

Demo iprange

binary.Uint32

```
func (bigEndian) Uint32(b []byte) uint32 {
    _ = b[3] // bounds check hint to compiler; see golang.org/issue/14808
    return uint32(b[3]) | uint32(b[2])<<8 | uint32(b[1])<<16 | uint32(b[0])<<24
}
```

We can test the panic:

```
// Small program to test panic when calling Uint32(nil).
package main

import (
    "encoding/binary"
)

func main() {
    _ = binary.BigEndian.Uint32(nil)
    // _ = binary.BigEndian.Uint32([]byte(nil))
}
```

Parse

```
case 5:
    ipDollar = ipS[ippt-3 : ippt+1]
    //line ip.y:54
    {
        mask := net.CIDRMask(int(ipDollar[3].num), 32)
        min := ipDollar[1].addrRange.Min.Mask(mask)
        maxInt := binary.BigEndian.Uint32([]byte(min)) + // <---- panic here
            0xffffffff -
            binary.BigEndian.Uint32([]byte(mask)) // <---- panic here
        maxBytes := make([]byte, 4)
        binary.BigEndian.PutUint32(maxBytes, maxInt)
        maxBytes = maxBytes[len(maxBytes)-4:]
        max := net.IP(maxBytes)
        ipVAL.addrRange = AddressRange{
            Min: min.To4(),
            Max: max.To4(),
        }
    }
```

CIDRMask

Returns `nil` if input is valid. For example `CIDRMask(80, 32)`.

```
// CIDRMask returns an IPMask consisting of `ones' 1 bits
// followed by 0s up to a total length of `bits' bits.
// For a mask of this form, CIDRMask is the inverse of IPMask.Size.
func CIDRMask(ones, bits int) IPMask {
    if bits != 8*IPv4len && bits != 8*IPv6len {
        return nil
    }
    if ones < 0 || ones > bits {
        return nil
    }
    // removed
}
```

Fix panic?

Check the value of mask:

- Before passing to CIDRMask.
- Immediately after reading input.

Why fix things? or philosophical rants

- Just finding vulns == limited value
- Pointing and laughing got us here today
- security `consultant` (what I am) ==> security `engineer` (what I want to be)

Fuzzing goexif2

goexif2 package

- https://github.com/xor-gate/goexif2
- Fork of https://github.com/rwcarlsen/goexif

goexif2

Provides decoding of basic exif and tiff encoded data. This project is a fork of `rwcarlsen/goexif` with many PR and patches integrated. Suggestions and pull requests are welcome.

Installation

To install the exif extraction cli tool, in a terminal type:

```
go install github.com/xor-gate/goexif2/cmd/goexif2
goexif2 <file>.jpg
```

Functionality is split into two packages - "exif" and "tiff" The exif package depends on the tiff package.

```
go get github.com/xor-gate/goexif2/exif
go get github.com/xor-gate/goexif2/tiff
```

goexif2 Fuzz.go

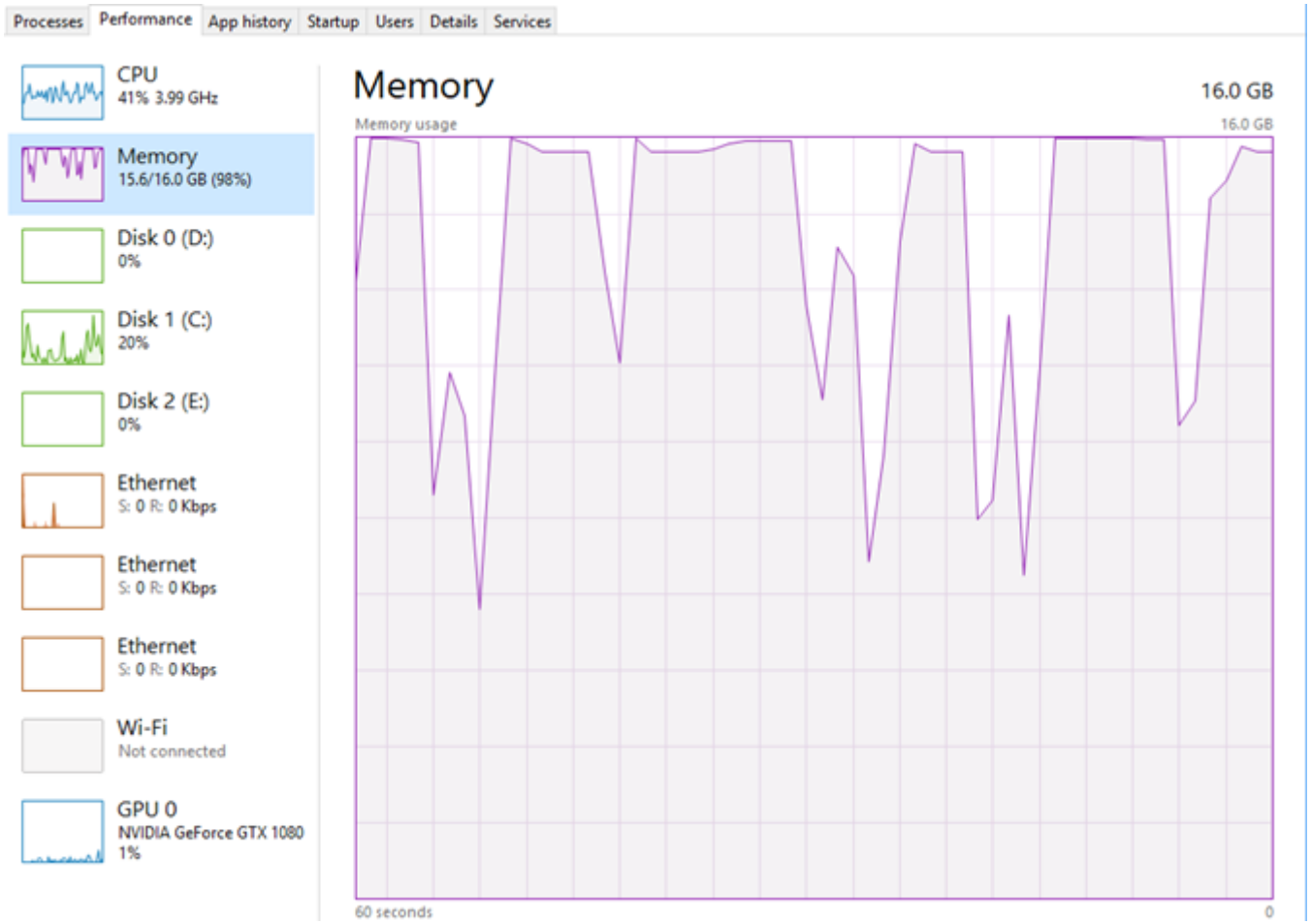
```
// +build gofuzz

package exif

import "bytes"

func Fuzz(data []byte) int {
    _, err := Decode(bytes.NewReader(data))
    if err != nil {
        return 0
    }
    return 1
}
```

Fuzzing problems



Out of memory crashes

These are usually false positives.

```
runtime: out of memory: cannot allocate 25769803776-byte block (25832882176 in use)
fatal error: out of memory

runtime stack:
runtime.throw(0x547da6, 0xd)
    /go-fuzz-build214414686/goroot/src/runtime/panic.go:616 +0x88
runtime.largeAlloc(0x600000000, 0x440001, 0x5f8330)
    /go-fuzz-build214414686/goroot/src/runtime/malloc.go:828 +0x117
runtime.mallocgc.func1()
    /go-fuzz-build214414686/goroot/src/runtime/malloc.go:721 +0x4d
runtime.systemstack(0x0)
    /go-fuzz-build214414686/goroot/src/runtime/asm_amd64.s:409 +0x7e
runtime.mstart()
    /go-fuzz-build214414686/goroot/src/runtime/proc.go:1175
```

Demo goexif2

Int64 crash

```
panic: runtime error: index out of range

goroutine 1 [running]:
github.com/xor-gate/goexif2/tiff.(*Tag).Int64(...)
    go-fuzz-build214414686/gopath/src/github.com/xor-gate/goexif2/tiff/tag.go:363
github.com/xor-gate/goexif2/exif.loadSubDir(0xc042080510, 0x547f15, 0xe, 0xc042080390, 0xc042080540, 0xc
    go-fuzz-build214414686/gopath/src/github.com/xor-gate/goexif2/exif/exif.go:211 +0x704
github.com/xor-gate/goexif2/exif.(*parser).Parse(0x613170, 0xc042080510, 0xc0420804b0, 0x0)
    go-fuzz-build214414686/gopath/src/github.com/xor-gate/goexif2/exif/exif.go:190 +0x174
github.com/xor-gate/goexif2/exif.Decode(0x560240, 0xc042080480, 0x5ae92f8f, 0x212abedc, 0x1e9999)
    go-fuzz-build214414686/gopath/src/github.com/xor-gate/goexif2/exif/exif.go:331 +0x503
github.com/xor-gate/goexif2/exif.Fuzz(0x38f0000, 0x72, 0x200000, 0xc042047f48)
    go-fuzz-build214414686/gopath/src/github.com/xor-gate/goexif2/exif/Fuzz.go:8 +0xba
go-fuzz-dep.Main(0x550580)
    go-fuzz-build214414686/goroot/src/go-fuzz-dep/main.go:49 +0xb4
main.main()
    go-fuzz-build214414686/gopath/src/github.com/xor-gate/goexif2/exif/go.fuzz.main/main.go:10 +0x34
exit status 2
```

tag.Int64 method

```
// Int64 returns the tag's i'th value as an integer. It returns an error if the
// tag's Format is not IntVal. It panics if i is out of range.
func (t *Tag) Int64(i int) (int64, error) {
    if t.format != IntVal {
        return 0, t.typeErr(IntVal)
    }
    return t.intVals[i], nil // <--- Panic
}
```

Fix it?

```
// Int64 returns the tag's i'th value as an integer. It returns an error if the
// tag's Format is not IntVal. It panics if i is out of range.
func (t *Tag) Int64(i int) (int64, error) {
    if t.format != IntVal {
        return 0, t.typeErr(IntVal)
    }
    if i >= len(t.intVals) {
        return 0, newTiffError("index out of range in intVals", nil)
    }
    return t.intVals[i], nil
}
```

More reading:

- https://github.com/dvyukov/go-fuzz
- https://medium.com/@dgryski/go-fuzz-github-com-arolek-ase-3c74d5a3150c
- https://mijailovic.net/2017/07/29/go-fuzz/
- https://blog.cloudflare.com/dns-parser-meet-go-fuzzer/
- https://go-talks.appspot.com/github.com/dvyukov/go-fuzz/slides/fuzzing.slide#1
- https://parsiya.net/blog/2018-04-29-learning-go-fuzz-1-iprange/
- https://parsiya.net/blog/2018-05-05-learning-go-fuzz-2-goexif2/

Thank you

Parsia