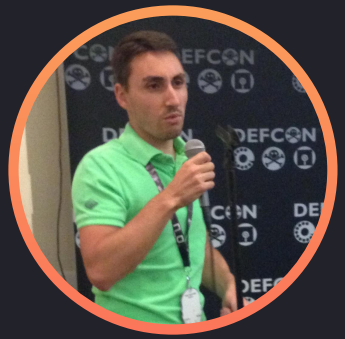
Getting more confident with your security helper libraries thanks to Go fuzzing

10 November 2022



Jeremy Matos

Principal Security Engineer

jmatosgrafana

Agenda

- Path traversal with Go
- Introduction to Go Fuzzing
- Fuzzing more complex code
- Writing predicates can be hard
- Next steps

Grafana: Our first 0-day in December 2021

- Responsibly disclosed by a researcher on December 2nd 2021 - path traversal in the Go code of Grafana: CVE-2021-43798
- Out of excitement, he tweeted about path traversal
- Actively exploited On December 7th, making it a 0-day
- Security fix released on December 7

But are we not protected by Go standard library?

filepath.Clean
is tricky

filepath.Clean is tricky

- Reading the doc too quickly:

  Clean returns the shortest path name equivalent to path

- The devil is in the details:

func Clean ¶

```
func Clean(path string) string
```

Clean returns the shortest path name equivalent to path by purely lexical processing. It applies the following rules iteratively until no further processing can be done:

1. Replace multiple Separator elements with a single one.
2. Eliminate each . path name element (the current directory).
3. Eliminate each inner .. path name element (the parent directory) along with the non-.. element that precedes it.
4. Eliminate .. elements that begin a rooted path: that is, replace "/.." by "/" at the beginning of a path, assuming Separator is '/'.

The returned path ends in a slash only if it represents a root directory, such as "/" on Unix or `C:\` on Windows.

Finally, any occurrences of slash are replaced by Separator.

If the result of this process is an empty string, Clean returns the string ".".

CONFUSING

filepath.Clean in simple words

- It removes any “..” sequence for
  - All the inner elements
  - The first element if it starts with /

- It **does not remove the first “..” element if it does not start with a /**


```
/data
../data

Program exited.
```

The vulnerable code

- Vulnerability

```
292         requestedFile := filepath.Clean(web.Params(c.Req)["*"])
293         pluginFilePath := filepath.Join(plugin.PluginDir, requestedFile)
```

- Interesting comment in code about a gosec warning:

  It's safe to ignore gosec warning G304 since we already clean the requested file path

- gosec warning G304
  - File path provided as taint input
  - The right way: use *filepath.Clean* !

The fixed code

- The corresponding PR
  - The fix

```
307        requestedFile := filepath.Clean(filepath.Join("/", web.Params(c.Req)["*"]))
```

  - Added 1 unit test

- Improvements discussed
  - Normalize URL in all routes
  - Silencing gosec rule (with the risk of not fixing the issue)
  - Security helper library

Introducing Go Fuzzing

Go Fuzzing

- Fuzzing in a few words
  - Extend unit tests by predicates describing “things that should never happen”
  - Generate many pseudo random inputs and test them against those predicates
- Available natively from Go 1.18
  - Identified violations trigger the creation of corresponding test data
  - Never ending loop (by default)
  - Multithreaded
- Rather than following the tutorial, let’s use the previous path traversal fix as example

Go Fuzzing example 1: validation logic

- Extracting the validation logic in a simple method
- Source code

```go
package cleanpath

import (
    "path/filepath"
)

func CleanPath(param string) string {
        return filepath.Clean(filepath.Join("/", param))
}
```

Go Fuzzing example 1: writing the predicates

- Writing the fuzzing test

```
8  func FuzzCleanPath(f *testing.F) {
9      testcases := []string {"README", "../../otherplugin/../README", ""}
10     for _, tc := range testcases {
11         f.Add(tc)
12     }
13     f.Fuzz(func(t *testing.T, param string) {
14         cleaned := CleanPath(param)
15
16         if !strings.HasPrefix(cleaned, "/") { //CleanPath should enforce that the string starts with a /
17           t.Errorf("Orginal input: %q, cleaned up: %q", param, cleaned)
18         }
19
20         if strings.Contains(cleaned, "/../") { //CleanPath should have removed all path traversal elements
21           t.Errorf("Orginal input: %q, cleaned up: %q", param, cleaned)
22         }
23     })
24 }
```

Go Fuzzing example 1: launch fuzzing

- Make sure that you have at least go 1.18
  - `go version`
- First validate that unit tests are passing
  - `go test`
- Start the fuzzing loop
  - `go test -fuzz=Fuzz`

Go Fuzzing example 1: fixing the predicates

- Trial and error when writing down the predicates
  - Fuzzing will find violations that are in fact valid outputs
- For some corner cases it will be hard to define if it is valid or invalid output
  - Fuzzing helps to make requirements more explicit
  - Less ambiguity, less vulnerabilities

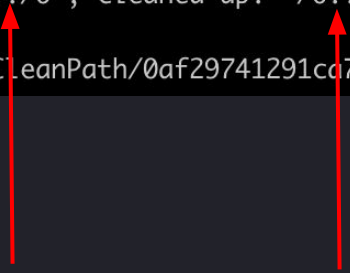
Go Fuzzing example 1: fixing the predicates

- E.g. changing the previous example with this new condition:

  `strings.Contains(cleaned, "../")`

```
--- FAIL: FuzzCleanPath (0.00s)
    cleanpath_test.go:21: Orginal input: "0../0", cleaned up: "/0../0"

Failing input written to testdata/fuzz/FuzzCleanPath/0af29741291ca701afa646cd35be722284688b77088390f5c3011c98bc19764e
```



legit input ?

valid output ?

Fuzzing more complex helpers

Go Fuzzing example 2: validation logic

- Source code used when checking signature of a Grafana plugin

```
9  // isSymlinkRelativeTo checks whether symlinkDestPath is relative to basePath.
10 // symlinkOrigPath is the path to file holding the symbolic link.
11 func isSymlinkRelativeTo(basePath string, symlinkDestPath string, symlinkOrigPath string) bool {
12   if filepath.IsAbs(symlinkDestPath) {
13     return false
14   } else {
15     fileDir := filepath.Dir(symlinkOrigPath)
16     cleanPath := filepath.Clean(filepath.Join(fileDir, "/", symlinkDestPath))
17     p, err := filepath.Rel(basePath, cleanPath)
18     if err != nil {
19       return false
20     }
21
22     if strings.HasPrefix(p, ".."+string(filepath.Separator)) {
23       return false
24     }
25   }
26
27   return true
28 }
```

Go Fuzzing example 2: abstracting the predicates

- Writing the [fuzzing test]

```
 9  func FuzzSymlinks(f *testing.F) {
10      testcases := []string {"README", "../otherplugin/README", "../otherplugin/../README"}
11      for _, tc := range testcases {
12          f.Add(tc)
13      }
14      f.Fuzz(func(t *testing.T, symlinkDestPath string) {
15          output := isSymlinkRelativeTo("/base", symlinkDestPath, "/base/plugins/symlink.txt")
16          expected := expectedResult("/base", symlinkDestPath, "/base/plugins/symlink.txt")
17
18          //testing output && !expected could be enough: not approving something that should not
19          if (output != expected) {
20            t.Errorf("Input: %q, Output: %t, Expected: %t", symlinkDestPath, output, expected)
21          }
22      })
23  }
```

Go Fuzzing example 2: writing the predicates

- Re-implementing some logic for the fuzzing test

```
25  func expectedResult(base string, destpath string, origpath string) bool {
26    if strings.HasPrefix(destpath, "/") {
27      return false //naive implementation instead of filePath.IsAbs
28    }
29
30    merged := filepath.Join(filepath.Dir(origpath),destpath)
31    if !strings.HasPrefix(merged, base) { //naive check of whether we stay in base folder
32      return false
33    }
34
35    return true
36  }
```

Go Fuzzing example 2: finding a corner case

- Launch fuzzing

```
--- FAIL: FuzzSymlinks (0.00s)
    relative_symlink_test.go:20: Input: "../..", Output: true, Expected: false
    Failing input written to testdata/fuzz/FuzzSymlinks/f959aa1c4f02[...]aab8
```

- `go test` will now fail with the added content in `testdata` folder
- Discussion about expected behavior in this Grafana PR
- The fix in the validator logic:

```
if strings.HasPrefix(p, ".."+string(filepath.Separator)) {
if p == ".." || strings.HasPrefix(p, ".."+string(filepath.Separator)) {
```

Writing predicates can be hard

Writing predicates can be challenging

- Fuzzing works best on small size helpers
  - Simple functions that have an easy to describe behaviour
  - More chance to have an obvious predicate implementation,
    e.g. ‘should not contain this character sequence’

- For medium size helpers, complex validation logic requires reimplementation
  - Copy pasting the original implementation in the fuzz test provides no value
  - Not getting biased by the original implementation

To which extent should standard libraries be trusted?

Lessons learned validating Grafana filestorage_api

- Re-implementing the rather complex `ValidatePath` function was time consuming
- Did not identify any violation
- Not 100% confident some corner cases have not been forgotten

Next steps

Next steps

- Make security helpers as simple as possible
- Include fuzzing in the CI/CD pipeline
- Communicate about those “trusted” security helpers
- Validate via semgrep rules that those helpers are indeed used

Key takeaways

- Beware of *filepath.Clean()* when protecting from path traversal

- Fuzzing is useful in real-life to:
  - Improve automated testing coverage
  - Identify corner cases that are not obvious

  thus allowing to become more confident with your security helper libraries

- Go Fuzzing is easy to use and efficient as long as you target simple functions

Thank you

Source code available at https://github.com/jmatosgrafana/gofuzzing


Grafana Labs