Wir bauen web fuzzer... ...in schnell

https://emile.space/events/2023/06-GPN/

Fuzzer, wasn' das?

Dirbuster, Wfuzz, Gobuster, Ffuf und Feroxbuster sind coole tools, aber was ist wenn ihr etwas spezielles habt? Wir bauen einen custom HTTP Fuzzer (der auch entsprechend schnell ist).

Vorraussetzungen:

– Laptop mit golang (https://go.dev) und python (https://www.python.org/downloads/) installiert.
– Setzt euch bitte ein golang Projekt auf und schaut das es funktioniert.

Golang

Setting up a project

```
Getting started
```

```
; mkdir fuzz
; cd fuzz
; go mod init github.com/<nick>/fuzz
```

```
Printing something (testing the setup)
```

```
// main.go
package main

import (
    "fmt"
)

func main() {
    fmt.Println("Hello World!")
}
```

```
// go run ./...
```

Making an HTTP request

```
package main

import (
        "fmt"
        "io"
        "net/http"
)

func main() {
        fmt.Println("hello world")

        resp, err := http.Get("https://emile.space")
        if err != nil {
                fmt.Println("Some error: ", err)
        }

        defer resp.Body.Close()

        body, err := io.ReadAll(resp.Body)
        if err != nil {
                fmt.Println("Some error: ", err)
        }

        fmt.Println(body)
}
```

```
The relevant parts of making an HTTP request
```

```
    resp, err := http.Get("http://example.com/")
    defer resp.Body.Close()
    if err != nil {
        // ...
    }

    body, err := io.ReadAll(resp.Body)
```

```
    if err != nil {
        // ...
    }

    fmt.Println(string(body))
```

Reading a file

```
import (
    "os"
    "strings"
)

...

    dat, err := os.ReadFile("./wordlist.txt")
    if err != nil {
        // ...
    }
    fmt.Print(string(dat))
```

Combine it!

```
        // read file
        dat, err := os.ReadFile("./wordlist.txt")
        if err != nil {
            fmt.Println("Some error: ", err)
        }

        words := strings.Split(string(dat), "\n")

        // loop over words
        for i, word := range words {
            fmt.Printf("http request nr. %d to word %s ", i, word)

            url := fmt.Sprintf("https://emile.space/%s", word)

            // make the request
            resp, err := http.Get(url)
            ...
```

SPEED

Cleanup: Functions!

```
func request(id int, url string) {
        resp, err := http.Get(url)
        if err != nil {
                fmt.Println("Some error: ", err)
        }

        defer resp.Body.Close()

        body, err := io.ReadAll(resp.Body)
        if err != nil {
                fmt.Println("Some error: ", err)
        }

        fmt.Printf("[%d] req to url %s (%d)\n", id, url, len(body))
}
```

"Channel"

https://gobyexample.com/channels

```
// https://gobyexample.com/channel-synchronization
package main

import (
    "fmt"
    "time"
)

func worker(done chan bool) {
    fmt.Print("working...")
    time.Sleep(time.Second)
    fmt.Println("done")
    done <- true
}
```

```
func main() {
    done := make(chan bool, 1)
    go worker(done)

    // something else could be doing stuff here at the same time as the
worker

    <-done
}
```

using this

```
...
func worker(wordChan chan string) {
    for {
        url := <-wordChan
        request(url)
    }
}
...

func main() {
    ...
    wordChan := make(chan string, 10)
    for i := 1; i < 4; i++ {
        go worker(wordChan)
    }
    ...
}
```

```
func worker(id int, wordChan chan string, doneChan chan bool) {
    out:
    for {
        select {

        case url := <-wordChan:
            url = fmt.Sprintf("https://emile.space/%s", url)
            request(id, url)
```

```
        case <-time.After(3 * time.Second):
            fmt.Printf("worker %d couldn't get a new url after 3 seconds,
quitting\n", id)
            break out

        }
    }
    doneChan <- true
}
```

```
func main() {
    // read wordlist
    dat, err := os.ReadFile("./wordlist1.txt")
    if err != nil {
        fmt.Println("Some error: ", err)
    }

    words := strings.Split(string(dat), "\n")

    // create workers
    wordChan := make(chan string, 10)
    doneChan := make(chan bool, 4)
    for i := 1; i < 4; i++ {
        go worker(i, wordChan, doneChan)
    }

    // fill word channel with all the words we want to fuzz
    fmt.Println("Filling wordChan")
    for _, word := range words {
        wordChan <- word
    }

    // check that all the workers are done before ending
    for i := 1; i < 4; i++ {
        <-doneChan
    }
}
```

```
    // create workers
    wordChan := make(chan string, 10)
    doneChan := make(chan bool, 4)
```

```
    for i := 1; i < 4; i++ {
        go worker(i, wordChan, doneChan)
    }

    // fill word channel with all the words we want to fuzz
    for _, word := range words {
        wordChan <- word
    }



    // check that all the workers are done before ending
    for i := 1; i < 4; i++ {
        <-doneChan
    }
```

https://emile.space/events/2023/04-easterhegg/

(Das war's vorerst)

Also, wir können einiges machen, um ein bisschen Struktur reinzubringen hier Aspekte die wir uns genauer anschauen werden:

In diesem Workshop werden wir einen benutzerdefinierten HTTP-Fuzzer entwickeln, der speziell auf deine Bedürfnisse abgestimmt ist und somit effektiver als allgemeine Tools. Wir werden die folgenden Aspekte berücksichtigen:

– Protokoll-Unterstützung: Wir stellen sicher, dass unser Fuzzer das HTTP-Protokoll vollständig unterstützt, einschließlich Methoden wie GET, POST, PUT usw.
– Geschwindigkeit: Geschwindigkeit ist ein wichtiger Faktor bei Fuzzing-Tools. Wir überlegen, wie wir die Geschwindigkeit optimieren können, z.B. durch parallele Verarbeitung und Verwendung effizienter Algorithmen.
– Payload-Generierung: Wir überlegen uns, wie wir Payloads automatisch generieren können, die auf deine spezifischen Bedürfnisse abgestimmt sind.
– Ergebnisanalyse: Wir stellen sicher, dass unser Fuzzer die Ergebnisse in einer leicht verständlichen Form bereitstellt, die es dir ermöglicht, Probleme schnell zu identifizieren und zu beheben.

Am Ende des Workshops hast du ein vollständiges Verständnis dafür, wie man einen benutzerdefinierten HTTP-Fuzzer entwickelt und eine eigene Version, die du für deine zukünftigen Sicherheitsbewertungen einsetzen kannst.