Introduction

Package goproxy provides a customizable HTTP proxy library for Go (golang),

It supports regular HTTP proxy, HTTPS through CONNECT, and "hijacking" HTTPS connection using "Man in the Middle" style attack.

The intent of the proxy, is to be usable with reasonable amount of traffic yet, customizable and programmable.

The proxy itself is simply a `net/http` handler.

In order to use goproxy, one should set their browser to use goproxy as an HTTP proxy. Here is how you do that in Chrome and in Firefox.

For example, the URL you should use as proxy when running `./bin/basic` is `localhost:8080`, as this is the default binding for the basic proxy.

Mailing List

New features would be discussed on the mailing list before their development.

Latest Stable Release

Get the latest goproxy from `gopkg.in/elazarl/goproxy.v1`.

Why not Fiddler2?

Fiddler is an excellent software with similar intent. However, Fiddler is not as customizable as goproxy intend to be. The main difference is, Fiddler is not intended to be used as a real proxy.

A possible use case that suits goproxy but not Fiddler, is, gathering statistics on page load times for a certain website over a week. With goproxy you could ask all your users to set their proxy to a dedicated machine running a goproxy server. Fiddler is a GUI app not designed to be ran like a server for multiple users.

A taste of goproxy

To get a taste of `goproxy`, a basic HTTP/HTTPS transparent proxy

```
package main

import (
    "github.com/elazarl/goproxy"
    "log"
    "net/http"
)

func main() {
```

```
    proxy := goproxy.NewProxyHttpServer()
    proxy.Verbose = true
    log.Fatal(http.ListenAndServe(":8080", proxy))
}
```

This line will add `X-GoProxy: yxorPoG-X` header to all requests sent through the proxy

```
proxy.OnRequest().DoFunc(
    func(r *http.Request,ctx *goproxy.ProxyCtx)(*http.Request,*http.Response) {
        r.Header.Set("X-GoProxy","yxorPoG-X")
        return r,nil
    })
```

`DoFunc` will process all incoming requests to the proxy. It will add a header to the request and return it. The proxy will send the modified request.

Note that we returned nil value as the response. Had we returned a response, goproxy would have discarded the request and sent the new response to the client.

In order to refuse connections to reddit at work time

```
proxy.OnRequest(goproxy.DstHostIs("www.reddit.com")).DoFunc(
    func(r *http.Request,ctx *goproxy.ProxyCtx)(*http.Request,*http.Response) {
        if h,_,_ := time.Now().Clock(); h >= 8 && h <= 17 {
            return r,goproxy.NewResponse(r,
                    goproxy.ContentTypeText,http.StatusForbidden,
                    "Don't waste your time!")
        }
        return r,nil
})
```

`DstHostIs` returns a `ReqCondition`, that is a function receiving a `Request` and returning a boolean we will only process requests that matches the condition. `DstHostIs("www.reddit.com")` will return a `ReqCondition` accepting only requests directed to "www.reddit.com".

`DoFunc` will receive a function that will preprocess the request. We can change the request, or return a response. If the time is between 8:00am and 17:00pm, we will neglect the request, and return a precanned text response saying "do not waste your time".

See additional examples in the examples directory.

What's New

1. Ability to `Hijack` CONNECT requests. See the eavesdropper example
2. Transparent proxy support for http/https including MITM certificate generation for TLS. See the transparent example.

License

I put the software temporarily under the Go-compatible BSD license, if this prevents someone from using the software, do let me know and I'll consider changing it.

At any rate, user feedback is very important for me, so I'll be delighted to know if you're using this package.

Beta Software

I've received a positive feedback from a few people who use goproxy in production settings. I believe it is good enough for usage.

I'll try to keep reasonable backwards compatibility. In case of a major API change, I'll change the import path.