```
  $.ajax("api/values", {
    type: "post",
    contentType: "application/json",
    data: { }, // JSON data goes here
    dataType: "json",
    headers: {
      'RequestVerificationToken': '@TokenHeaderValue()'
    }
  });
</script>
```

7.12 GO

Go è un linguaggio di programmazione open source, sviluppato da Google e pubblicato per la prima volta nel 2009. È nato dall'esigenza di avere un linguaggio facile da imparare, specializzato nella programmazione concorrente e che avesse un compilatore in grado di produrre eseguibili efficienti e veloci. La sintassi è molto simile al C.

7.12.1 Client Dom Stored XSS

Come riconoscerla

Cross-site scripting (XSS) è una vulnerabilità che affligge siti web dinamici che impiegano un insufficiente controllo dell'input, in qualsiasi modo pervenuto. Un attacco di XSS permette a un malintenzionato di inserire o eseguire codice lato client al fine di attuare un insieme variegato di operazioni quali ad esempio: raccolta, manipolazione e reindirizzamento di informazioni riservate, visualizzazione e modifica di dati presenti sui server, alterazione del comportamento dinamico delle pagine web ecc.

GO, proprio come qualsiasi altro linguaggio di programmazione multiuso, è vulnerabile a XSS nonostante la documentazione indirizzi chiaramente sull'utilizzo di html/template package.

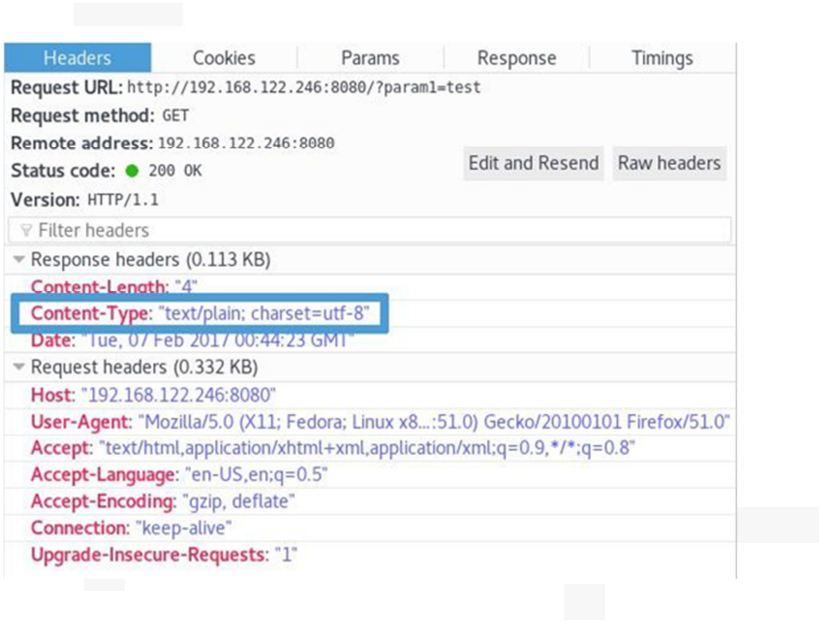
In riferimento al seguente frammento di codice:

```
package main
import "net/http"
import "io"
func handler (w http.ResponseWriter, r
    *http.Request) { io.WriteString(w,
    r.URL.Query().Get("param1"))
}
func main () {
    http.HandleFunc("/", handler)
    http.ListenAndServe(":8080", nil)
}
```

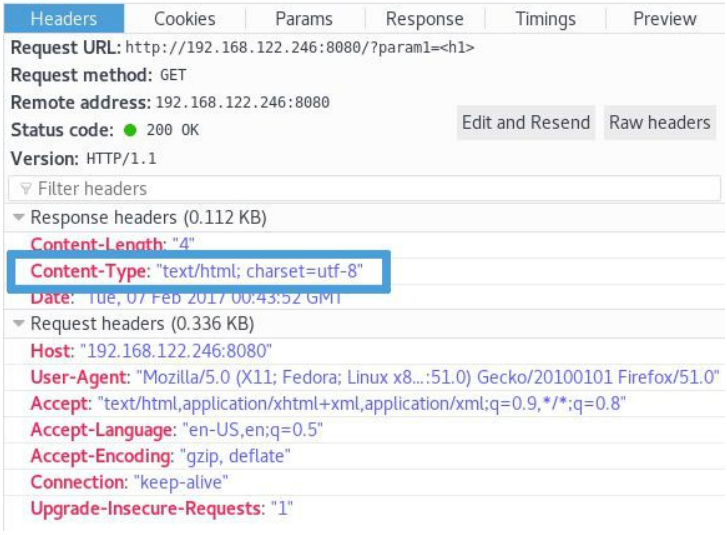
Questo codice crea e avvia un server HTTP in ascolto sulla porta 8080 (main()) gestendo le richieste sulla root del server (/).

La funzione handler(), che gestisce le richieste, prevede un parametro query stringa Param1, il cui valore viene quindi scritto nel flusso di risposta (w):

Se param1=test, il Content-Type sarà inviato come text/plain:

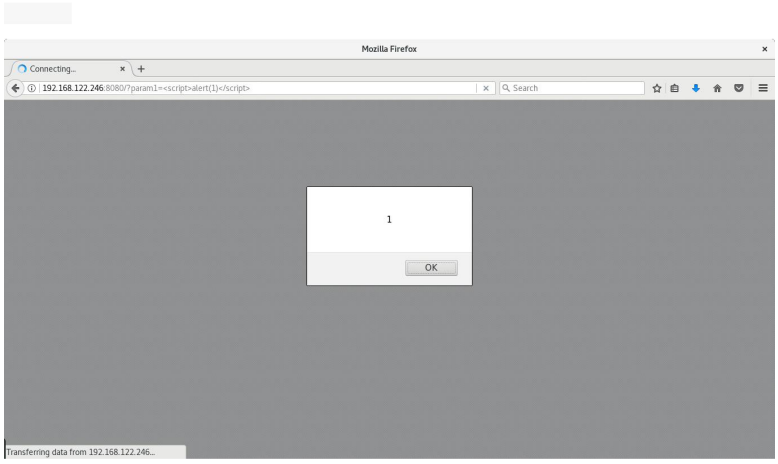


Se param1=<h1>, il Content-Type sarà inviato come text/html (ciò rende vulnerabile a XSS):



Si potrebbe pensare che rendere param1 uguale a qualsiasi tag HTML porti allo stesso comportamento, ma non è così: param1=<h2>, param1=, param1=<form> non modificano Content-Type in text/html, bensì in plain / text.

Se param1=<script>alert(1)</script>, il Content-Type sarà inviato come text/html e il valore sarà restituito e quindi facilmente interpretato tramite l'alert (XSS - Cross Site Scripting):



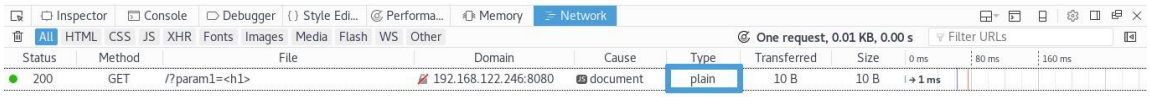
Come difendersi

Sostituire il text/template package con html/template:

```
package main
import "net/http"
import "html/template"
func handler(w http.ResponseWriter, r *http.Request)
        { param1 := r.URL.Query().Get("param1")
        tmpl := template.New("hello")
        tmpl, _ = tmpl.Parse(`{{define "T"}}{{.}}{{end}}`)
        tmpl.ExecuteTemplate(w, "T", param1)
}

func main() {
        http.HandleFunc("/", handler)
        http.ListenAndServe(":8080", nil)
}
```

Se `param1=<h1>`, l'intestazione di risposta HTTP `Content-Type` non verrà inviata come `text/plain` :



`Param1` è correttamente codificato sul browser: