Code Source – Tom / Ilies

- **main.go**

```
package main

import (
	"fmt"
	"network-scanner/reports"
	"network-scanner/scanner"
	"strconv"
)

func main() {
	var results []scanner.ScanResult
	for i := 1; i <= 10; i++ {
		ip := "10.49.34." + strconv.Itoa(i)
		fmt.Printf("Scanning %s...\n", ip)
		result := scanner.ScanIP(ip, 1, 1024)
		results = append(results, result)
	}
	// Export des résultats
	reports.ExportToJSON(results, "network-
scanner/results/scan_results.json")
	reports.ExportToHTML(results, "network-
scanner/results/scan_results.html")
	fmt.Println("Scan terminé.")
}
```

- **scanner/scan_ip.go**

```
package scanner

import (
	"fmt"
	"net"
	"time"
)

// ScanIP scanne une adresse IP et détecte les ports ouverts
func ScanIP(ip string, startPort, endPort int) ScanResult {
	var result ScanResult
	result.IP = ip
	timeout := 500 * time.Millisecond
	for port := startPort; port <= endPort; port++ {
		address := fmt.Sprintf("%s:%d", ip, port)
		conn, err := net.DialTimeout("tcp", address, timeout)
		if err == nil {
			conn.Close()
			result.Ports = append(result.Ports, OpenPort{
				Port:    port,
				Service: "Unknown",
				CVEs:    nil,
			})
		}
	}
	return result
}
```

- **scanner/port_scanner.go**

```
package scanner

// ScanResult stocke les résultats d'un scan sur une adresse IP

type ScanResult struct {

	IP    string     `json:"ip"`

	Ports []OpenPort `json:"ports"`

}

// OpenPort contient les informations sur un port ouvert

type OpenPort struct {

	Port    int      `json:"port"`

	Service string   `json:"service"`

	CVEs    []string `json:"cves"`

}
```

- **scanner/service_identifier.go**

```
package scanner

// GetServiceName retourne le nom du service correspondant à un
port connu

func GetServiceName(port int) string {

	services := map[int]string{

		21: "FTP", 22: "SSH", 23: "Telnet", 25: "SMTP",

		53: "DNS", 80: "HTTP", 443: "HTTPS", 3306: "MySQL",

	}

	if name, exists := services[port]; exists {

		return name

	}

	return "Unknown"

}
```

- **reports/scan_results_json.go**

```
package reports

import (
	"encoding/json"
	"fmt"
	"network-scanner/scanner"
	"os"
	"path/filepath"
)

func ExportToJSON(results []scanner.ScanResult, filename string) error
{
	// Créer le répertoire parent si nécessaire
	if err := os.MkdirAll(filepath.Dir(filename), os.ModePerm); err
!= nil {
		return fmt.Errorf("failed to create directory: %v", err)
	}
	data, err := json.MarshalIndent(results, "", "  ")
	if err != nil {
		return fmt.Errorf("failed to marshal JSON: %v", err)
	}
	if err := os.WriteFile(filename, data, 0644); err != nil {
		return fmt.Errorf("failed to write file: %v", err)
	}
	return nil
}
```

- **reports/scan_results_html.go**

```
package reports

import (
    "os"
    "network-scanner/scanner"
    "fmt"
)

func ExportToHTML(results []scanner.ScanResult, filename string) error
{
    file, err := os.Create(filename)
    if err != nil {
        return err
    }
    defer file.Close()

    file.WriteString("<html><body><h1>Scan Results</h1><ul>")
    for _, result := range results {
        file.WriteString(fmt.Sprintf("<li>%s: %v</li>", result.IP,
result.Ports))
    }
    file.WriteString("</ul></body></html>")

    return nil
}
```

- **vulnerability/cve_database.go**

```
package vulnerability

import (

      "encoding/json"

      "fmt"

      "net/http"

      "net/url"

      "time"

)

type CVE struct {

      ID          string `json:"id"`

      Description string `json:"description"`
```

```
}
type nvdResponse struct {
	Vulnerabilities []struct {
		CVE struct {
			ID          string `json:"id"`
			Description struct {
				DescriptionData []struct {
					Value string `json:"value"`
				} `json:"description_data"`
			} `json:"description"`
		} `json:"cve"`
	} `json:"result"`
}
func FetchCVE(service string) []CVE {
	var cves []CVE
	// Encodage approprié du paramètre de recherche
	encodedService := url.QueryEscape(service)
	baseURL :=
"https://services.nvd.nist.gov/rest/json/cves/2.0?keywordSearch="
+ encodedService
	client := &http.Client{Timeout: 10 * time.Second}
	req, err := http.NewRequest("GET", baseURL, nil)
	if err != nil {
		fmt.Printf("Error creating request: %v\n", err)
		return cves
	}
	// Ajout des en-têtes requis
```

```
	req.Header.Add("User-Agent", "NetworkScanner/1.0")
	resp, err := client.Do(req)
	if err != nil {
		fmt.Printf("Error fetching CVEs: %v\n", err)
		return cves
	}
	defer resp.Body.Close()
	if resp.StatusCode != http.StatusOK {
		fmt.Printf("API returned status code: %d\n",
resp.StatusCode)
		return cves
	}
	var result nvdResponse
	if err := json.NewDecoder(resp.Body).Decode(&result); err
!= nil {
		fmt.Printf("Error parsing JSON: %v\n", err)
		return cves
	}
	for _, vuln := range result.Vulnerabilities {
		if len(vuln.CVE.Description.DescriptionData) > 0 {
			cves = append(cves, CVE{
				ID:          vuln.CVE.ID,
				Description:
vuln.CVE.Description.DescriptionData[0].Value,
			})
		}
	}
	return cves   }
```