Project Documentation

Part 1 : Subdomain Enumeration Tool Documentation

1. Overview

The Subdomain Enumeration Tool is designed to automate the process of discovering subdomains for a given domain using various public APIs. The tool integrates with services such as CRT.sh, URLScan.io, VirusTotal, and SecurityTrails to gather subdomain data. It is built using the Go programming language and is the first part of a larger project aimed at simplifying reconnaissance tasks for cybersecurity purposes.

2. Features

- **API Integrations**: Fetches subdomains from multiple services:
  - CRT.sh
  - URLScan.io
  - VirusTotal
  - SecurityTrails
- **Duplicate Removal**: Ensures that the subdomains collected from different services are unique and sorted.
- **Output File**: Saves the enumerated subdomains into a specified output file (`recon_enum.txt` by default).
- **Dependency Check**: Verifies the availability of required external tools (e.g., curl, jq, httpx).

---

3. Prerequisites

- **Go 1.18+**
- **External Dependencies**:
  - `curl`
  - `jq`

  - `httpx`
  - `httprobe`

Make sure these tools are installed on the system before running the program.

---

4. Setup Instructions

1. **Install Go Dependencies**:
   Install Go if not already installed:

   ```
   sudo apt install golang-go  # For Ubuntu
   brew install go             # For macOS
   ```

2. **Obtain API Keys**:
   Get API keys from the following services:

   - CRT.sh (no API key required)
   - URLScan.io
   - VirusTotal
   - SecurityTrails

   Update the following variables in the code with your API keys:

   ```
   CERTSPOTTER_API_KEY     = "YOUR_CERTSPOTTER_API_KEY"
   VIRUSTOTAL_API_KEY      = "YOUR_VIRUSTOTAL_API_KEY"
   SECURITYTRAILS_API_KEY = "YOUR_SECURITYTRAILS_API_KEY"
   URLSCAN_API_KEY         = "YOUR_URLSCAN_API_KEY"
   ```

---

5. Usage

The tool can be run in two modes:

1. **Single Domain Mode**: Specify a domain using the `d` flag.

   ```
   go run gohack.go -d example.com
   ```

2. **File Input Mode**: Use the `f` flag to specify a file containing multiple domains.

```
go run gohack.go -f domains.txt
```

3. **Output File**: The default output file is `recon_enum.txt`. You can specify a custom file using the `o` flag.

```
go run gohack.go -d example.com -o output.txt
```

---

6. Tool Components

6.1. API Enumerations

- **CRT.sh Enumeration**:
  The tool queries
  CRT.sh to retrieve a list of subdomains associated with the given domain by looking at certificate transparency logs.

  Example code:

  ```
  func enumerateCRTSh(domain string) []string {
      // Code to fetch data from CRT.sh
  }
  ```

- **URLScan.io Enumeration**:
  Uses the
  URLScan.io API to search for subdomains related to the domain in question.

  Example code:

  ```
  func enumerateUrlscan(domain string) ([]string, error) {
      // Code to fetch data from URLScan.io
  }
  ```

- **VirusTotal Enumeration**:
  Retrieves subdomains by querying VirusTotal's database of domains and IP addresses.

  Example code:

```
func enumerateVirusTotal(domain string) []string {
    // Code to fetch data from VirusTotal
}
```

- **SecurityTrails Enumeration**:
  Fetches subdomain data from SecurityTrails' extensive domain information database.

  Example code:

  ```
  func enumerateSecurityTrails(domain string) []string {
      // Code to fetch data from SecurityTrails
  }
  ```

6.2. Unique Subdomain Sorting

Ensures that subdomains collected from different sources are unique and sorted alphabetically before being saved to the output file.

```
func uniqueSorted(subdomains []string) []string {
    // Code to remove duplicates and sort
}
```

7. Error Handling

The tool includes error handling for various scenarios, such as:

- Failure to connect to the API.
- Invalid JSON responses.
- File I/O errors when reading or writing the output.

Example:

```
if err != nil {
    log.Fatalf("Error: %v", err)
}
```

8. Example Output

When the tool completes the enumeration, the output file contains a sorted list of unique subdomains:

```
blog.example.com
mail.example.com
shop.example.com
```

The output will vary depending on the APIs used and the data they provide.

9. Future Improvements

- Add more subdomain enumeration APIs (e.g., Shodan, Censys).
- Implement concurrency to improve performance.
- Add more flexible output formats (e.g., JSON, CSV).

Part 2: NahamStore

> 💡 This part is explained in depth in Report file

1. Overview

In this part, the focus is on identifying and exploiting vulnerabilities in the NahamStore web application. This challenge is from TryHackMe and simulates a real-world environment where web applications can have common security flaws. The goal is to perform reconnaissance, discover vulnerabilities, and exploit them to gain access to sensitive information or unauthorized functionality.

2. Prerequisites

Before starting the vulnerability discovery, ensure that you have:

- **Basic knowledge of web vulnerabilities** (e.g., SQL injection, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF)).
- **Tools Installed**:
    - Burp Suite for intercepting and analyzing HTTP requests.

- Nmap (for scanning open ports and services).
- Gobuster or Dirbuster (for directory enumeration).
- SQLmap (for SQL injection testing).

3. Step-by-Step Vulnerability Discovery Process

3.1. Reconnaissance and Information Gathering

1. **Nmap Scan**: Begin with an Nmap scan to discover open ports, services, and versions running on the target.

   ```
   nmap -A -T4 <target-ip>
   ```

2. **Directory Enumeration**: Use Gobuster to enumerate directories and hidden files on the web server.

   ```
   gobuster dir -u http://<target-ip> -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt
   ```

3. **Service and Version Information**: Identify the CMS, web server, or any frameworks in use (e.g., WordPress). This helps in looking for known vulnerabilities (CVE).

3.2. Vulnerability Discovery

1. **Identify Login Forms or Input Fields**: Look for pages that contain forms where user input is required (login, registration, search bars).
2. **Test for SQL Injection (SQLi)**: Use SQLmap or manual SQL injection techniques to check if the application is vulnerable to SQLi attacks.
   - **Example**: Use a common SQL injection payload (`' OR 1=1 --`) in login forms or input fields.
   - Run SQLmap:

   ```
   sqlmap -u "http://<target-ip>/login" --forms --dbs
   ```

3. **Cross-Site Scripting (XSS)**: Check if input fields are vulnerable to XSS by injecting JavaScript payloads like:

```
   <script>alert('XSS');</script>
   ```

   Analyze if the input is being reflected in the response without proper sanitization.

4. **Cross-Site Request Forgery (CSRF)**: Investigate if sensitive actions (like changing account details) can be performed by forging a request from a logged-in user.
5. **File Upload Vulnerabilities**: Test if the file upload functionality accepts executable files or allows the upload of malicious scripts.

3.3. Exploitation

Once vulnerabilities have been discovered, the next step is to exploit them. In NahamStore, common vulnerabilities might include SQL injection, XSS, and privilege escalation.

1. **SQL Injection**:
   - If the login form is vulnerable to SQLi, bypass authentication by injecting a payload like `' OR 1=1 --` into the username and password fields.
   - Gain access to admin functionalities and sensitive data from the database.
2. **XSS Attack**:
   - If an XSS vulnerability is present, inject a script into a vulnerable form and trigger it to steal session cookies or perform actions as another user.
3. **CSRF Exploitation**:
   - If CSRF vulnerabilities exist, create a malicious HTML form that tricks the user into submitting a request to change their account password or make unauthorized transactions.

---

4. Example: NahamStore Vulnerability Exploitation

In the NahamStore challenge, the following steps outline how vulnerabilities were discovered and exploited:

1. **SQL Injection in Login**:
   - By testing with a basic SQL injection payload in the login form, we were able to bypass authentication and access the admin panel.
   - Payload: `' OR 1=1 --`
2. **Directory Traversal**:
   - The `gobuster` scan revealed a hidden directory `/admin/` that contained sensitive configuration files.
   - By exploiting a directory traversal vulnerability, we accessed the `/etc/passwd` file and obtained information about system users.
3. **Cross-Site Scripting**:
   - A search field in the NahamStore app was vulnerable to XSS. Injecting the following script resulted in an alert pop-up:

     ```
     <script>alert('XSS in NahamStore');</script>
     ```
4. **Privilege Escalation**:
   - By analyzing the application's roles, we found that regular users could escalate their privileges by modifying their profile information through insecure API endpoints.

5. Tools Used

- **Burp Suite**: Intercepted and analyzed HTTP requests and responses to look for vulnerabilities in form submissions.
- **SQLmap**: Automated SQL injection testing and exploitation.
- **Gobuster**: Discovered hidden directories and files on the web server.
- **Nikto**: Basic web server scanning to detect outdated software and misconfigurations.