CSE 406 Project Report

GOBUSTER : A Content Discovery Tool

Report Prepared by:

Abrar Mahmud - 1905097
Shahriar Raj - 1905105

Supervised by:

A.K.M Mehedi Hasan

Department of CSE, BUET

Contents

1 Introduction

Gobuster is a powerful, open-source tool designed to enumerate files and directories on web/application servers. It is written in Go, making it capable of doing high-performance work. Gobuster is commonly used by cybersecurity professionals, including penetration testers and ethical hackers, to discover hidden resources within web servers and DNS structures that are not typically visible or linked from the main pages of a website.

The tool operates by using wordlists that contain numerous filenames, directory names, or subdomains. These wordlists are used as the basis to systematically check for the existence of resources on the target server or domain. Gobuster can identify potentially unsecured files, directories, and subdomains that might expose sensitive information or reveal insights about the backend structure of the web application, thus highlighting areas that may require further investigation or immediate security measures. The tool is mainly Linux-based.

1.1 Key Features of Gobuster

- **Directory/File Enumeration:** Quickly identifies hidden or unlisted directories and files on web servers by brute-forcing URIs using wordlists.
- **DNS Subdomain Enumeration:** Discovers subdomains by brute-forcing domain name systems, helping to map out a target's DNS structure.
- **Virtual Host Scanning:** Can identify virtual hosts configured on web servers.
- **Support for Various Protocols:** Works with HTTP, HTTPS, and supports other schemes by integrating with proxy servers.
- **Customizable:** Supports various flags and configurations to customize scans, including setting custom headers, using cookies, and ignoring SSL/TLS certificate warnings.
- **Concurrent Processing:** Utilizes the Go language's powerful concurrency features to perform multiple requests or queries simultaneously, significantly speeding up the enumeration process.
- **Flexible:** Allows users to exclude certain response sizes, specify status codes to find or ignore, and even resume interrupted scans.

1.2 Why use Gobuster?

- It is free and OpenSource
- It is fast and easy to run
- Can be used with customized wordlists
- Supports various protocols beyond HTTP/HTTPS like FTP

Gobuster is popular in the cybersecurity community for its simplicity, speed, and effectiveness. It's an essential tool in the vulnerability assessment and penetration testing processes, aiding in the early stages of assessment by providing insights into possible points of entry and areas requiring deeper security analysis. Whether used for educational purposes, ethical hacking, or professional cybersecurity assessments, Gobuster serves as a critical component in the toolkit of modern security practitioners, emphasizing the importance of proactive security measures and thorough digital infrastructure examination.

2 Installation

Pre Requirements for Gobuster Installation:

- Linux (Preferably Kali Linux)
- go (atleast version 1.16)

Installing go:

```
sudo apt update
sudo apt install golang-go
```

Installing gobuster:

```
sudo apt install gobuster
```

3 Source Code Overview

Gobuster is a content discovery tool written in **go** language. The source code link can be found in the following link: [1]

Each mode of operation is implemented in a different folder in gobuster. Short description of each part is given below:

3.1 cli

The cli directory is used for parsing the command line argument and showing output to the console or to a file. If we look at the file **gobuster.go** in the cli folder, we can find some goroutine functions.

- **resultWorker:** It listens for results from **libgobuster.Gobuster** on a channel, formats the results into a string, and prints them to the console or to an output file.
- **errorWorker:** It listens for errors from **libgobuster.Gobuster** on a channel and prints the errors.
- **messageWorker:** It listens for log messages from **libgobuster.Gobuster** on a channel and prints the log messages based on their levels(debug, error and info).

- **progressWorker:** It uses a ticker to print the progress at regular intervals.

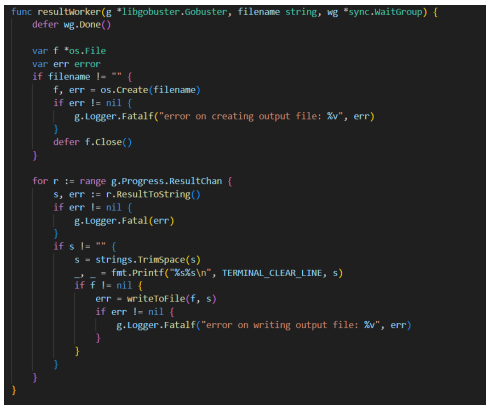


Figure 1: resultWorker Function

Figure 2: errorWorker and messageWorker Function

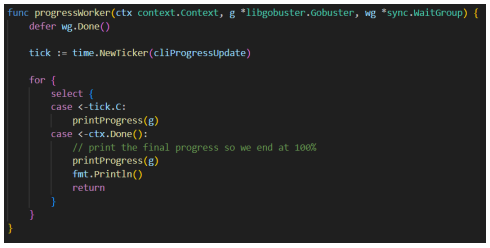


Figure 3: progressWorker Function

The **Gobuster** function is the main entry point of the CLI. It sets up the environment, initializes **libgobuster.Gobuster**, and starts goroutines for workers. It first prints the Gobuster version and configuration details. Then it runs the Gobuster scan with the given context, waits for it to complete, and cleans up after.

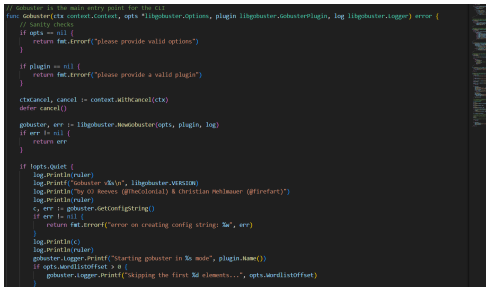


Figure 4: Gobuster Function

3.2 libgobuster

The libgobuster directory declares the basic interface of our gobuster object. We can see the basic interface of the gobuster object in the libgobuster.go file of the current folder.

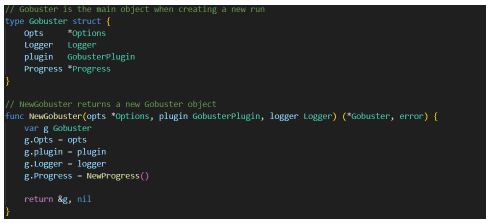


Figure 5: Structure of a Gobuster Object

The opts parameter defines the mode of operation and the related flags of that mode. Each mode of operation extends this struct to show their outputs.

Figure 6: Run function of a Gobuster Object

The **Run** function creates a word channel, takes words as inputs and then uses the **worker** function. In the worker function, the **processWord** function does all mode specific operations. All modes implement this function differently.

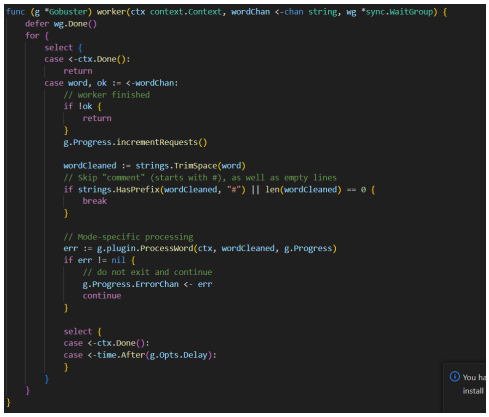


Figure 7: Worker function of a Gobuster Object

Besides, in the **http.go** file in this folder, an http connection is set up to send and receive data from the target url.

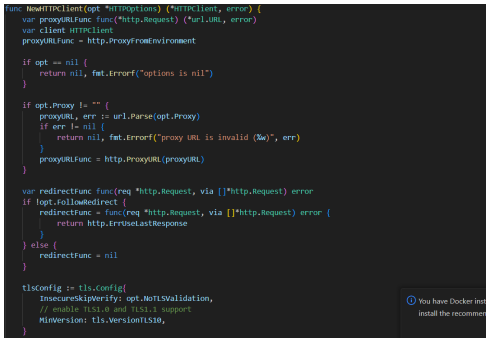


Figure 8: Setting up an http connection

3.3 gobusterdir

This mode is used to find hidden directories in a website. If we go to the file **gobusterdir.go** and view the **processWord** function, then we can see that it does the following:

- It constructs the full URL by appending the given word from the word list to the base URL, like **example.com/word** format. It ensures that the base URL ends with a slash and removes any leading slashes from the word.
- It allows for multiple attempts in case of timeout errors based on the values of the flags.

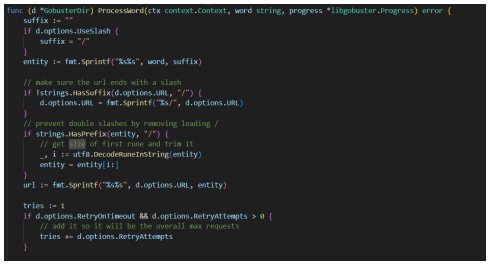


Figure 9: Constructing URL in Directory mode

- Then the function sends an http request using **d.http.Request**.
- As a result, we get status code, size of response from the return value, and the result is sent to the result channel.

Figure 10: Sending an http request in directory mode

3.4 gobusterdns

The DNS mode is used to find subdomains of a website. In DNS mode, the tool appends words in front of the domain name and sends DNS queries to verify which subdomains exist. If we go to the file **gobusterdns.go** and view the **processWord** function, then we can see that it does the following:

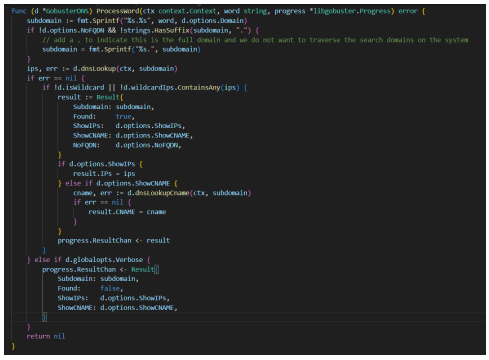


Figure 11: processWord function in DNS mode

- It constructs the subdomain by appending the current word to the specified Domain, like **word.example.com** format.
- It performs a DNS lookup for the constructed subdomain using the **dnsLookup** method. If there is no error, it checks for wildcards and ensures the subdomain is not part of the wildcard IPs.
- If **ShowIPs** option is enabled, it includes the resolved IPs in the result. Then it sends the results to the result channel.
- For each word in the wordlist, it either reports the successful resolution of the subdomain or, if in **Verbose** mode, reports when a subdomain is not found.

3.5 gobustervhost

For looking for sites that are hosted virtually by the same website, we send http requests to the same website with different hostname in header. If we go to the file **gobuster-vhost.go** and view the **processWord** function, then we can see that it does the following:

- First it constructs the subdomain name from the given word file.

Figure 12: Constructing Subdomain in vhost mode

- Then it sends an http request to the subdomain and gets response body, response size and status code as a response.

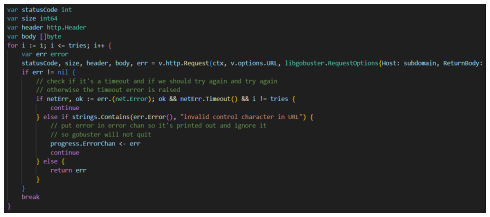


Figure 13: Get Response in vhost mode

- Now we need to see if the response body is the same as default response body or error response body. So, we send an http request to the domain name to get **normal body**. We also send abnormal request using a **uuid generator** and save the response body as **abnormal body**.
- Finally, we compare our response with normal and abnormal body. If there's no match with either, we can be sure that the response body is from a different website. Thus, we get a new website hosted by the base domain.

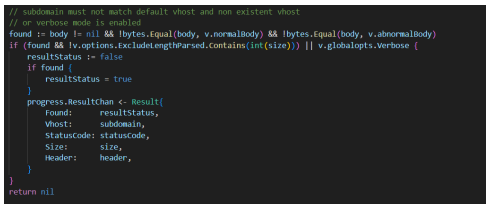


Figure 14: Compare Responses in vhost mode

3.6 gobusterfuzz

Fuzzing is the process of finding different parameters on a website. In FUZZ mode, we can insert content from wordlists into different places of a website and check the responses to understand which parameters are valid for that website. If we go to the file **gobusterfuzz.go** and view the **processWord** function, then we can see that it does the following:

- First, the function replaces the **FuzzKeyword** (a placeholder for the fuzzed payload) in the target URL with the current word from the wordlist.

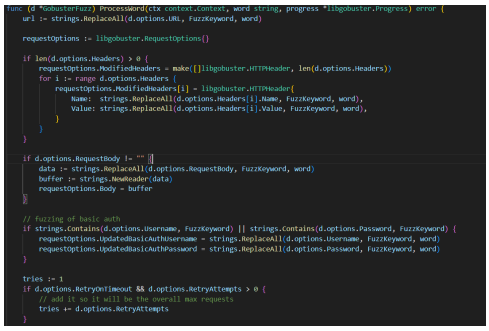


Figure 15: Constructing URL in fuzz mode

- Then it sends an http request to the target URL.

Figure 16: Sending request in fuzz mode

- We get the status code, response size and body. Upon examining these, we can determine which parameters are valid for the website.

3.7 gobustergcs

This mode is used to find publicly available gcs buckets. If we go to the file **gobustergcs.go** and view the **processWord** function, then we can see that it does the following:

- First, the function constructs the target URL with the current word from the wordlist, like this **https://storage.googleapis.com/storage/v1/b/bucket/o** format.
- Then it sends an http request to the target URL.

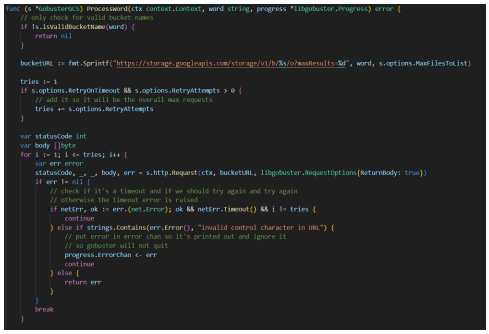


Figure 17: Constructing URL and getting response in gcs mode

- We get the status code, response size and body. From there after clicking on the media links, we can find the publicly available gcs buckets.

3.8 gobusters3

This mode works almost identically to gcs. The difference is in the url that is required to access public S3 buckets of amazon. If we go to the file **gobusters3.go** and view the **processWord** function, then we can see that it does the following:

- First, the function constructs the target URL with the current word from the wordlist, like this **https://bucket.s3.amazonaws.com/** format.
- Then it sends an http request to the target URL.

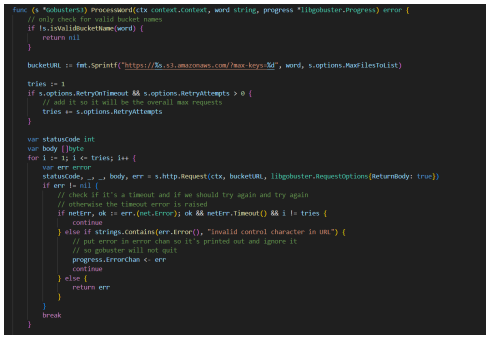


Figure 18: Constructing URL and getting response in s3 mode

- We get the status code, response size and body. From there we can visit the links and see which files are accessible.

3.9 gobustertftp

Tftp servers are a type of ftp servers, but with less protection. Tftp servers don't require any password to access them. We can use brute forcing of gobuster to find some filenames in a particular tftp server. If we go to the file **gobustertftp.go** and view the **processWord** function, then we can see that it does the following:

- First, the client is connected to the given server address.

- Then we enumerate our word file for words and send an http request to the server to find the target file with the same name there.

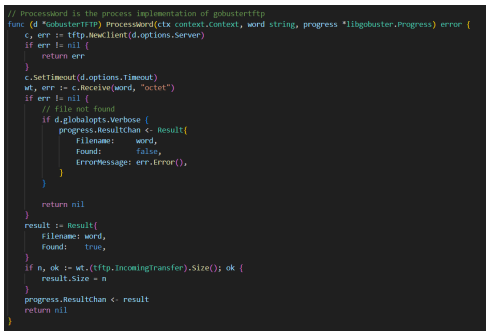


Figure 19: Connecting to server and getting response in tftp mode

- As tftp does not have any protection, we can get the files by the same name returned. Thus, we can find the files in our tftp server.

4 Documentation

Running the features in gobuster is very straightforward. We just need to run the commands for the specific mode in gobuster. We can see the basic modes of operation using the **gobuster –help** command.

```
$ gobuster --help
Usage:
  gobuster [command]

Available Commands:
  completion  Generate the autocompletion script for the
   specified shell
  dir         Uses directory/file enumeration mode
  dns         Uses DNS subdomain enumeration mode
  fuzz        Uses fuzzing mode. Replaces the keyword FUZZ in the
    URL, Headers and the request body
  gcs         Uses gcs bucket enumeration mode
  help        Help about any command
  s3          Uses aws bucket enumeration mode
  tftp        Uses TFTP enumeration mode
  version     shows the current version
  vhost       Uses VHOST enumeration mode (you most probably want
    to use the IP address as the URL parameter)

Flags:
      --debug                   Enable debug output
      --delay duration          Time each thread waits between
   requests (e.g. 1500ms)
  -h, --help                    help for gobuster
      --no-color                Disable color output
```

```
      --no-error              Don't display errors
  -z, --no-progress           Don't display progress
  -o, --output string         Output file to write results to (
   defaults to stdout)
  -p, --pattern string        File containing replacement
   patterns
  -q, --quiet                 Don't print the banner and other
   noise
  -t, --threads int           Number of concurrent threads (
   default 10)
  -v, --verbose               Verbose output (errors)
  -w, --wordlist string       Path to the wordlist. Set to - to
   use STDIN.
      --wordlist-offset int   Resume from a given position in the
    wordlist (defaults to 0)
```

4.1 DIR

The basic structure of dir mode is given below:

```
gobuster dir -u https://example.com -w parameters.txt
```

- dir: DIR mode of operation
- -u: The URL string
- -w: Path to the wordlist file

We can find more about the flags using the command **gobuster dir –help**

```
$ gobuster dir --help
Uses directory/file enumeration mode

Usage:
  gobuster dir [flags]

Flags:
  -f, --add-slash                          Append / to each
   request
      --client-cert-p12 string             a p12 file to use for
   options TLS client certificates
      --client-cert-p12-password string    the password to the p12
    file
      --client-cert-pem string             public key in PEM
   format for optional TLS client certificates
      --client-cert-pem-key string         private key in PEM
   format for optional TLS client certificates (this key needs to
    have no password)
  -c, --cookies string                     Cookies to use for the
   requests
  -d, --discover-backup                    Also search for backup
   files by appending multiple backup extensions
```

```
      --exclude-length string             exclude the following
   content lengths (completely ignores the status). You can
   separate multiple lengths by comma and it also supports ranges
    like 203-206
  -e, --expanded                          Expanded mode, print
   full URLs
  -x, --extensions string                 File extension(s) to
   search for
  -X, --extensions-file string            Read file extension(s)
   to search from the file
  -r, --follow-redirect                   Follow redirects
  -H, --headers stringArray               Specify HTTP headers, -
   H 'Header1: val1' -H 'Header2: val2'
  -h, --help                              help for dir
      --hide-length                       Hide the length of the
   body in the output
  -m, --method string                     Use the following HTTP
   method (default "GET")
      --no-canonicalize-headers           Do not canonicalize
   HTTP header names. If set header names are sent as is.
  -n, --no-status                         Don't print status
   codes
  -k, --no-tls-validation                 Skip TLS certificate
   verification
  -P, --password string                   Password for Basic Auth
      --proxy string                      Proxy to use for
   requests [http(s)://host:port] or [socks5://host:port]
      --random-agent                      Use a random User-Agent
    string
      --retry                             Should retry on request
    timeout
      --retry-attempts int                Times to retry on
   request timeout (default 3)
  -s, --status-codes string               Positive status codes (
   will be overwritten with status-codes-blacklist if set). Can
   also handle ranges like 200,300-400,404.
  -b, --status-codes-blacklist string     Negative status codes (
   will override status-codes if set). Can also handle ranges
   like 200,300-400,404. (default "404")
      --timeout duration                  HTTP Timeout (default
   10s)
  -u, --url string                        The target URL
  -a, --useragent string                  Set the User-Agent
   string (default "gobuster/3.6")
  -U, --username string                   Username for Basic Auth

Global Flags:
      --debug                 Enable debug output
      --delay duration        Time each thread waits between
   requests (e.g. 1500ms)
      --no-color              Disable color output
```

```
      --no-error               Don't display errors
  -z, --no-progress            Don't display progress
  -o, --output string          Output file to write results to (
   defaults to stdout)
  -p, --pattern string         File containing replacement
   patterns
  -q, --quiet                  Don't print the banner and other
   noise
  -t, --threads int            Number of concurrent threads (
   default 10)
  -v, --verbose                Verbose output (errors)
  -w, --wordlist string        Path to the wordlist. Set to - to
   use STDIN.
      --wordlist-offset int    Resume from a given position in the
    wordlist (defaults to 0)
```

4.2 DNS

The basic structure of dns mode is given below:

```
gobuster dns -d example.com -w subdomains.txt
```

- dns: DNS mode of operation
- -d: The base URL string
- -w: Path to the wordlist file

We can find more about the flags using the command **gobuster dns –help**

```
$ gobuster dns --help
Uses DNS subdomain enumeration mode

Usage:
  gobuster dns [flags]

Flags:
  -d, --domain string      The target domain
  -h, --help               help for dns
      --no-fqdn            Do not automatically add a trailing
   dot to the domain, so the resolver uses the DNS search domain
  -r, --resolver string    Use custom DNS server (format server.
   com or server.com:port)
  -c, --show-cname         Show CNAME records (cannot be used
   with '-i' option)
  -i, --show-ips           Show IP addresses
      --timeout duration   DNS resolver timeout (default 1s)
      --wildcard           Force continued operation when
   wildcard found

Global Flags:
      --debug                  Enable debug output
```

```
      --delay duration         Time each thread waits between
   requests (e.g. 1500ms)
      --no-color               Disable color output
      --no-error               Don't display errors
  -z, --no-progress            Don't display progress
  -o, --output string          Output file to write results to (
   defaults to stdout)
  -p, --pattern string         File containing replacement
   patterns
  -q, --quiet                  Don't print the banner and other
   noise
  -t, --threads int            Number of concurrent threads (
   default 10)
  -v, --verbose                Verbose output (errors)
  -w, --wordlist string        Path to the wordlist. Set to - to
   use STDIN.
      --wordlist-offset int    Resume from a given position in the
    wordlist (defaults to 0)
```

4.3 VHOST

The basic structure of vhost mode is given below:

```
gobuster vhost -u https://example.com -w words.txt
```

- vhost: VHOST mode of operation
- -u: The base URL string
- -w: Path to the wordlist file

We can find more about the flags using the command **gobuster vhost –help**

```
$ gobuster vhost --help
Uses VHOST enumeration mode (you most probably want to use the IP
    address as the URL parameter)

Usage:
  gobuster vhost [flags]

Flags:
      --append-domain                       Append main domain from
    URL to words from wordlist. Otherwise the fully qualified
   domains need to be specified in the wordlist.
      --client-cert-p12 string              a p12 file to use for
   options TLS client certificates
      --client-cert-p12-password string     the password to the p12
    file
      --client-cert-pem string              public key in PEM
   format for optional TLS client certificates
      --client-cert-pem-key string          private key in PEM
   format for optional TLS client certificates (this key needs to
    have no password)
```

```
  -c, --cookies string                     Cookies to use for the
   requests
      --domain string                      the domain to append
   when using an IP address as URL. If left empty and you specify
    a domain based URL the hostname from the URL is extracted
      --exclude-length string              exclude the following
   content lengths (completely ignores the status). You can
   separate multiple lengths by comma and it also supports ranges
    like 203-206
  -r, --follow-redirect                    Follow redirects
  -H, --headers stringArray                Specify HTTP headers, -
   H 'Header1: val1' -H 'Header2: val2'
  -h, --help                               help for vhost
  -m, --method string                      Use the following HTTP
   method (default "GET")
      --no-canonicalize-headers            Do not canonicalize
   HTTP header names. If set header names are sent as is.
  -k, --no-tls-validation                  Skip TLS certificate
   verification
  -P, --password string                    Password for Basic Auth
      --proxy string                       Proxy to use for
   requests [http(s)://host:port] or [socks5://host:port]
      --random-agent                       Use a random User-Agent
    string
      --retry                              Should retry on request
    timeout
      --retry-attempts int                 Times to retry on
   request timeout (default 3)
      --timeout duration                   HTTP Timeout (default
   10s)
  -u, --url string                         The target URL
  -a, --useragent string                   Set the User-Agent
   string (default "gobuster/3.6")
  -U, --username string                    Username for Basic Auth

Global Flags:
      --debug                 Enable debug output
      --delay duration        Time each thread waits between
   requests (e.g. 1500ms)
      --no-color              Disable color output
      --no-error              Don't display errors
  -z, --no-progress           Don't display progress
  -o, --output string         Output file to write results to (
   defaults to stdout)
  -p, --pattern string        File containing replacement
   patterns
  -q, --quiet                 Don't print the banner and other
   noise
  -t, --threads int           Number of concurrent threads (
   default 10)
  -v, --verbose               Verbose output (errors)
```

```
  -w, --wordlist string       Path to the wordlist. Set to - to
   use STDIN.
      --wordlist-offset int   Resume from a given position in the
    wordlist (defaults to 0)
```

4.4 FUZZ

The basic structure of fuzz mode is given below:

```
gobuster fuzz -u https://example.com?FUZZ=test -w parameters.txt
```

- fuzz: FUZZ mode of operation
- -u: The URL string
- -w: Path to the wordlist file

We can find more about the flags using the command **gobuster fuzz –help**

```
$ gobuster fuzz --help
Uses fuzzing mode. Replaces the keyword FUZZ in the URL, Headers
   and the request body

Usage:
  gobuster fuzz [flags]

Flags:
  -B, --body string                          Request body
      --client-cert-p12 string               a p12 file to use for
   options TLS client certificates
      --client-cert-p12-password string      the password to the p12
    file
      --client-cert-pem string               public key in PEM
   format for optional TLS client certificates
      --client-cert-pem-key string           private key in PEM
   format for optional TLS client certificates (this key needs to
    have no password)
  -c, --cookies string                       Cookies to use for the
   requests
      --exclude-length string                exclude the following
   content lengths (completely ignores the status). You can
   separate multiple lengths by comma and it also supports ranges
    like 203-206
  -b, --excludestatuscodes string            Excluded status codes.
   Can also handle ranges like 200,300-400,404.
  -r, --follow-redirect                      Follow redirects
  -H, --headers stringArray                  Specify HTTP headers, -
   H 'Header1: val1' -H 'Header2: val2'
  -h, --help                                 help for fuzz
  -m, --method string                        Use the following HTTP
   method (default "GET")
```

```
      --no-canonicalize-headers           Do not canonicalize
   HTTP header names. If set header names are sent as is.
  -k, --no-tls-validation                 Skip TLS certificate
   verification
  -P, --password string                   Password for Basic Auth
      --proxy string                      Proxy to use for
   requests [http(s)://host:port] or [socks5://host:port]
      --random-agent                      Use a random User-Agent
    string
      --retry                             Should retry on request
   timeout
      --retry-attempts int                Times to retry on
   request timeout (default 3)
      --timeout duration                  HTTP Timeout (default
   10s)
  -u, --url string                        The target URL
  -a, --useragent string                  Set the User-Agent
   string (default "gobuster/3.6")
  -U, --username string                   Username for Basic Auth

Global Flags:
      --debug                 Enable debug output
      --delay duration        Time each thread waits between
   requests (e.g. 1500ms)
      --no-color              Disable color output
      --no-error              Don't display errors
  -z, --no-progress           Don't display progress
  -o, --output string         Output file to write results to (
   defaults to stdout)
  -p, --pattern string        File containing replacement
   patterns
  -q, --quiet                 Don't print the banner and other
   noise
  -t, --threads int           Number of concurrent threads (
   default 10)
  -v, --verbose               Verbose output (errors)
  -w, --wordlist string       Path to the wordlist. Set to - to
   use STDIN.
      --wordlist-offset int   Resume from a given position in the
    wordlist (defaults to 0)
```

4.5 GCS

The basic structure of gcs mode is given below:

```
gobuster gcs -w bucket-names.txt
```

- gcs: gcs mode of operation
- -w: Path to the wordlist file

We can find more about the flags using the command **gobuster gcs –help**

```
$ gobuster gcs --help
Uses gcs bucket enumeration mode

Usage:
  gobuster gcs [flags]

Flags:
      --client-cert-p12 string            a p12 file to use for
   options TLS client certificates
      --client-cert-p12-password string   the password to the p12
    file
      --client-cert-pem string            public key in PEM
   format for optional TLS client certificates
      --client-cert-pem-key string        private key in PEM
   format for optional TLS client certificates (this key needs to
    have no password)
  -h, --help                              help for gcs
  -m, --maxfiles int                      max files to list when
   listing buckets (only shown in verbose mode) (default 5)
  -k, --no-tls-validation                 Skip TLS certificate
   verification
      --proxy string                      Proxy to use for
   requests [http(s)://host:port] or [socks5://host:port]
      --random-agent                      Use a random User-Agent
    string
      --retry                             Should retry on request
    timeout
      --retry-attempts int                Times to retry on
   request timeout (default 3)
      --timeout duration                  HTTP Timeout (default
   10s)
  -a, --useragent string                  Set the User-Agent
   string (default "gobuster/3.6")

Global Flags:
      --debug                 Enable debug output
      --delay duration        Time each thread waits between
   requests (e.g. 1500ms)
      --no-color              Disable color output
      --no-error              Don't display errors
  -z, --no-progress           Don't display progress
  -o, --output string         Output file to write results to (
   defaults to stdout)
  -p, --pattern string        File containing replacement
   patterns
  -q, --quiet                 Don't print the banner and other
   noise
  -t, --threads int           Number of concurrent threads (
   default 10)
  -v, --verbose               Verbose output (errors)
  -w, --wordlist string       Path to the wordlist. Set to - to
```

```
   use STDIN.
      --wordlist-offset int   Resume from a given position in the
    wordlist (defaults to 0)
```

4.6 S3

The basic structure of s3 mode is given below:

```
gobuster s3 -w bucket-names.txt
```

- s3: s3 mode of operation
- -w: Path to the wordlist file

We can find more about the flags using the command **gobuster s3 –help**

```
$ gobuster s3 --help
Uses aws bucket enumeration mode

Usage:
  gobuster s3 [flags]

Flags:
      --client-cert-p12 string            a p12 file to use for
   options TLS client certificates
      --client-cert-p12-password string   the password to the p12
    file
      --client-cert-pem string            public key in PEM
   format for optional TLS client certificates
      --client-cert-pem-key string        private key in PEM
   format for optional TLS client certificates (this key needs to
    have no password)
  -h, --help                              help for s3
  -m, --maxfiles int                      max files to list when
   listing buckets (only shown in verbose mode) (default 5)
  -k, --no-tls-validation                 Skip TLS certificate
   verification
      --proxy string                      Proxy to use for
   requests [http(s)://host:port] or [socks5://host:port]
      --random-agent                      Use a random User-Agent
    string
      --retry                             Should retry on request
    timeout
      --retry-attempts int                Times to retry on
   request timeout (default 3)
      --timeout duration                  HTTP Timeout (default
   10s)
  -a, --useragent string                  Set the User-Agent
   string (default "gobuster/3.6")

Global Flags:
      --debug                 Enable debug output
```

```
      --delay duration         Time each thread waits between
   requests (e.g. 1500ms)
      --no-color               Disable color output
      --no-error               Don't display errors
  -z, --no-progress            Don't display progress
  -o, --output string          Output file to write results to (
   defaults to stdout)
  -p, --pattern string         File containing replacement
   patterns
  -q, --quiet                  Don't print the banner and other
   noise
  -t, --threads int            Number of concurrent threads (
   default 10)
  -v, --verbose                Verbose output (errors)
  -w, --wordlist string        Path to the wordlist. Set to - to
   use STDIN.
      --wordlist-offset int    Resume from a given position in the
    wordlist (defaults to 0)
```

4.7 TFTP

The basic structure of tftp mode is given below:

```
gobuster tftp -s tftp.example.com -w common-filenames.txt
```

- tftp: tftp mode of operation
- -s: The target tftp server
- -w: Path to the wordlist file

We can find more about the flags using the command **gobuster tftp –help**

```
$ gobuster tftp --help
Uses TFTP enumeration mode

Usage:
  gobuster tftp [flags]

Flags:
  -h, --help               help for tftp
  -s, --server string      The target TFTP server
      --timeout duration   TFTP timeout (default 1s)

Global Flags:
      --debug                  Enable debug output
      --delay duration         Time each thread waits between
   requests (e.g. 1500ms)
      --no-color               Disable color output
      --no-error               Don't display errors
  -z, --no-progress            Don't display progress
```

```
-o, --output string          Output file to write results to (
 defaults to stdout)
-p, --pattern string         File containing replacement
 patterns
-q, --quiet                  Don't print the banner and other
 noise
-t, --threads int            Number of concurrent threads (
 default 10)
-v, --verbose                Verbose output (errors)
-w, --wordlist string        Path to the wordlist. Set to - to
 use STDIN.
    --wordlist-offset int    Resume from a given position in the
  wordlist (defaults to 0)
```

5 Demonstration

We will see some demonstration of the 7 different modes of gobuster.

5.1 DIR

In the directory mode, we scan for hidden directories and vulnerable files in the websites. A demonstration of directory busting: **First Example:**

```
$ gobuster dir -u https://cse.buet.ac.bd -w test_wordlist.txt -t
   25
```

Here we are searching for vulnerable files in cse.buet.ac.bd. The output files are below:

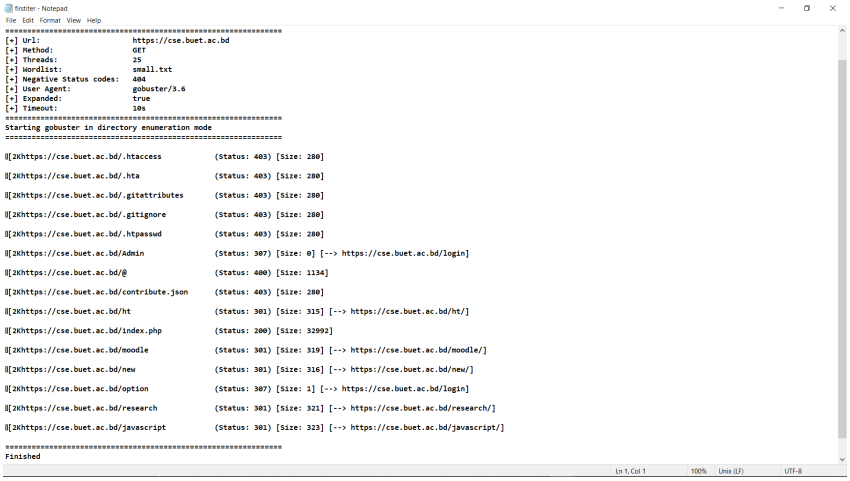


Figure 20: Output of the dir mode on cse.buet.ac.bd

We get cse.buet.ac.bd/javascript. If we further scan cse.buet.ac.bd/javascript, we get the following output:

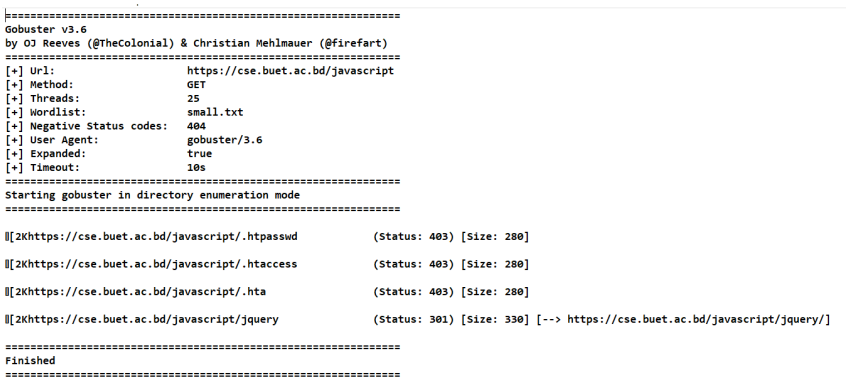


Figure 21: Output of the dir mode on cse.buet.ac.bd/javascript

If we scan further on cse.buet.ac.bd/javascript/jquery, then we get the following output:

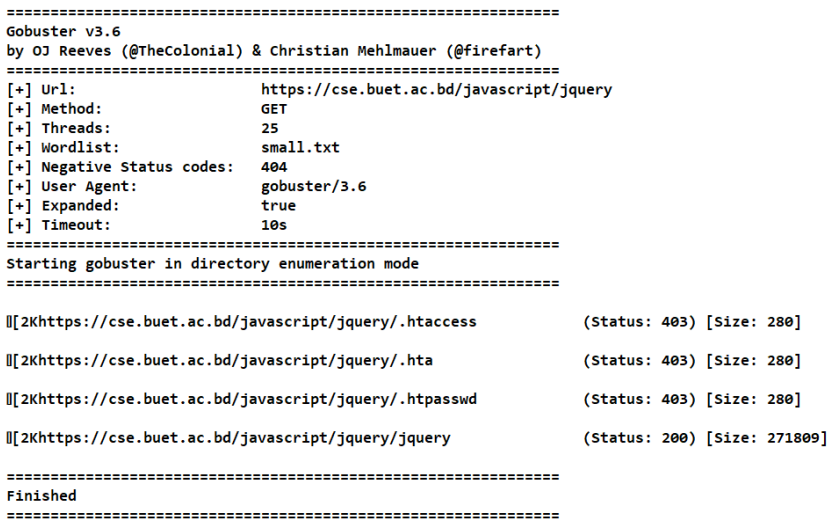


Figure 22: Output of the dir mode on cse.buet.ac.bd/javascript/jquery

We got a hidden file cse.buet.ac.bd/javascript/jquery/jquery which can be accessed. If we access the url, we get:

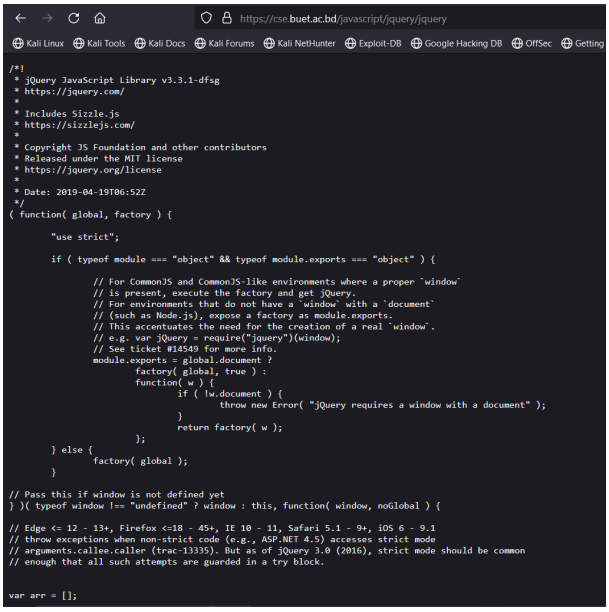


Figure 23: Accessing cse.buet.ac.bd/javascript/jquery/jquery

Second Example: We will now solve a ctf problem using gobuster. We can find the problem statement here: Problem

We can get a look at the problem too.

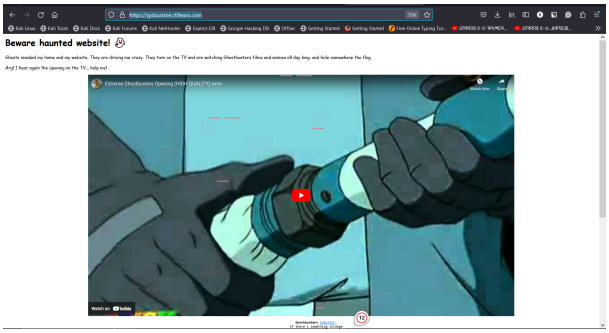


Figure 24: Problem Statement of gobustme

For solving the problem we scan the website using directory busting. The given Command:

```
gobuster dir -u https://gobustme.ctflearn.com -w common1.txt
```

The output for this is as follows:

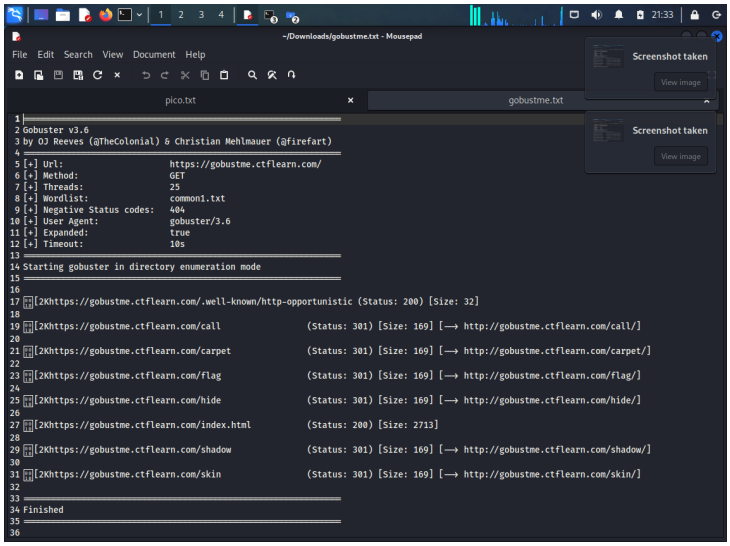


Figure 25: output of scanning gobustme

We can access Flag hidden Site and get the flag.

The flag: CTFlearngh0sbu5t3rs$_4$*ever*

5.2 DNS

By dns busting, we can discover the subdomains of a site. We can again take cse.buet.ac.bd as example.

The command for the discovery:

```
$ gobuster dns -d cse.buet.ac.bd -w subdomains-top1million-5000.
   txt -t 25
```

The discovery:

```
===============================================================
Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Domain:     cse.buet.ac.bd
[+] Threads:    25
[+] Timeout:    1s
[+] Wordlist:   subdomains-top1million-5000.txt
===============================================================
Starting gobuster in DNS enumeration mode
===============================================================

[2KFound: moodle.cse.buet.ac.bd

[2KFound: ra.cse.buet.ac.bd

===============================================================
Finished
===============================================================
```

We have found 2 subdomains of cse.buet.ac.bd. One is moodle and another is ra.

5.3 VHOST

Now using vhost mode we will try to access the subsdomains.
The Command:

```
$ gobuster vhost -u  google.com -w subdomains-top1million-5000.
   txt -t 25 --append-domain
```

The Output:

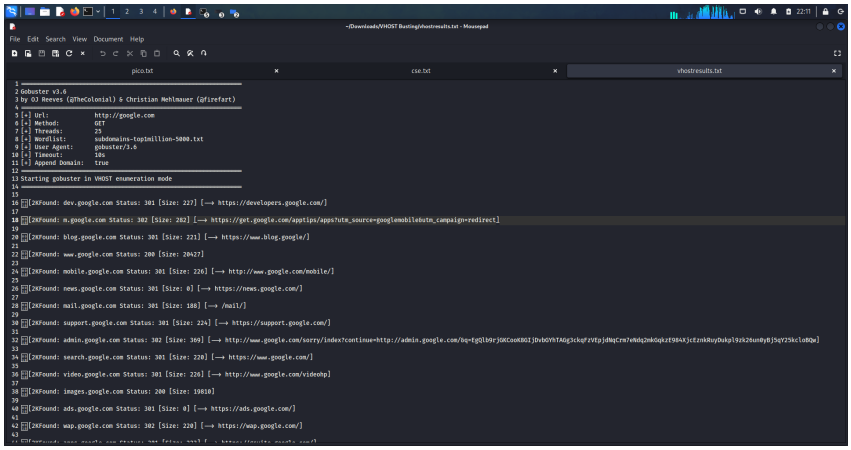


Figure 26: Output of the vhost mode on google.com

5.4 FUZZ

In this mode, we replace the word **FUZZ** by a word of the word file each time and find valid parameters for the url. We are using a word file from Seclists from this link[3] For demonstration, we are using the following command:

```
gobuster fuzz -u https://play.picoctf.org/practice?FUZZ=1 -w /
   home/piyal/SecLists/Discovery/Web-Content/BurpSuite-ParamMiner
   /lowercase-headers -b 400,301 > fuzz2.txt
```

Different parts of the command are explained below:

- fuzz: fuzz mode of operation
- -u: The target url, with the word **FUZZ**, that will be replaced by a word from the word file
- -w: Path to the wordlist file
- -b: The status code 400 and 301 are blocked, so we will not get any response with status code 400 or 301
- : fuzz2.txt: The output will be written to fuzz2.txt file

The output is shown below:

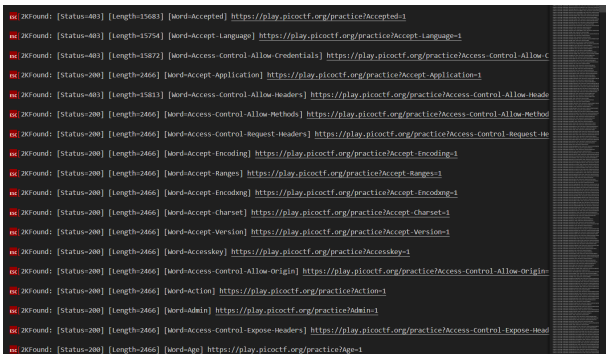


Figure 27: Output of the fuzz mode

We can select a link from the file and view the contents of the link:

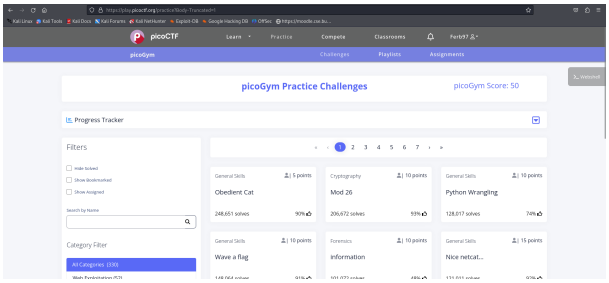


Figure 28: Output result of the fuzz mode

Here, the **FUZZ** got replaced by the word **truncated**, and we can see the results. This means **truncated** is a valid parameter to the website. The attackers can use this mode to find valid parameters to launch attacks by injecting values in the parameters.

5.5 GCS

GCS mode finds the publicly available buckets from google cloud storage for a given word file. For demonstration, we are using the text file **s3words.txt**. We run the following command:

```
gobuster gcs -w s3words.txt -t 10 > gcs1.txt
```

Different parts of the command are explained below:

- gcs: gcs mode of operation
- -w: Path to the wordlist file
- -t: number of threads
- : gcs1.txt: The output will be written to gcs1.txt file

The output is shown below:

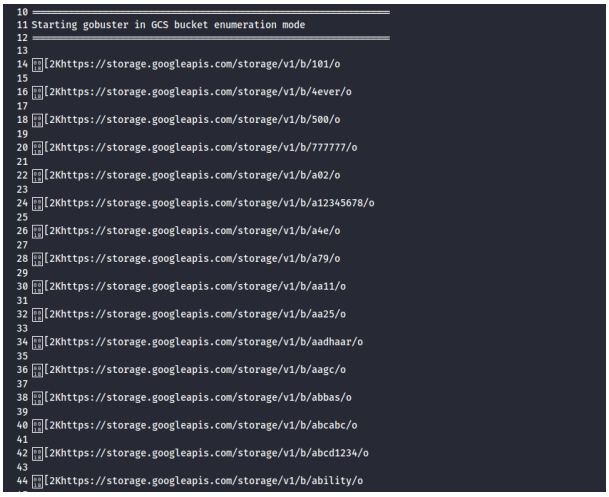


Figure 29: Output of the gcs mode

If we paste this link, we will get an output like the following:

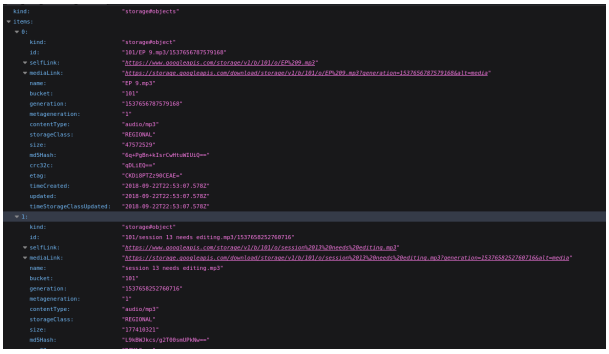


Figure 30: Output Media Link of the gcs mode

The **mediaLink** part contains the link for the file. All files are not accessible here. So, if we want to access the links, we may get a status 403 code.

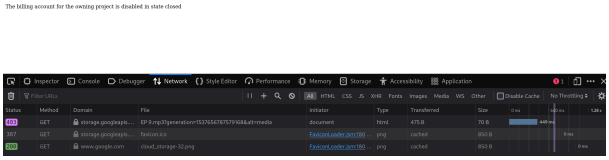


Figure 31: Failed Output of the gcs mode

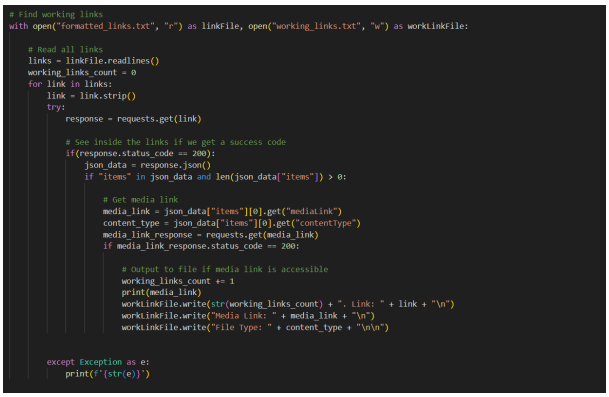
To find out which links are accessible, we write a python file named **find_working_links.py**. The function of the python file is explained below:

- It reads the output from **gcs1.txt** file.
- Then it formats the so they can be used to find files.
- The formatted links are then written to **formatted_links.txt** file.
- Now, we read the **formatted_links.txt** file and send request.

- If the status code of the response is 200, we extract the **mediaLink** part and send request to know if the media exists.
- If we get a status response 200, we write the **link**, **mediaLink** and **File type** to **working_links.txt** file.
- Thus we get all the working media links in **working_links.txt** file.

```python
import requests
import re

# Read from output file
with open("gcs1.txt", "r") as txt:
    string = txt.read()

# Format links
links = re.findall(r'https://[^\s]+', string)

# Write to output file
with open("formatted_links.txt", "w") as output:
    for link in links:
        output.write(link+"\n")
```

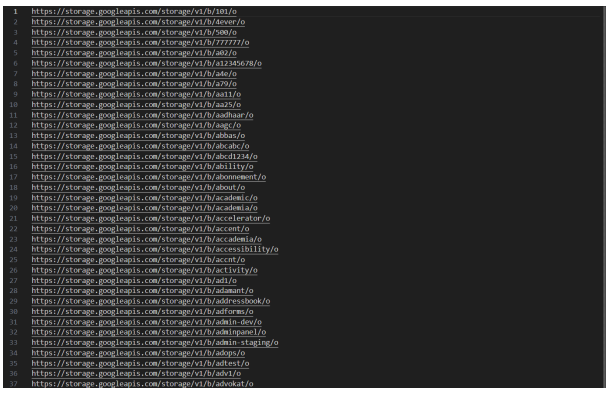
Figure 32: Code for formatting the gcs output



```python
# Find working links
with open("formatted_links.txt", "r") as linkFile, open("working_links.txt", "w") as workLinkFile:

    # Read all links
    links = linkFile.readlines()
    working_links_count = 0
    for link in links:
        link = link.strip()
        try:
            response = requests.get(link)

            # See inside the links if we get a success code
            if(response.status_code == 200):
                json_data = response.json()
                if "items" in json_data and len(json_data["items"]) > 0:

                    # Get media link
                    media_link = json_data["items"][0].get("mediaLink")
                    content_type = json_data["items"][0].get("contentType")
                    media_link_response = requests.get(media_link)
                    if media_link_response.status_code == 200:

                        # Output to file if media link is accessible
                        working_links_count += 1
                        print(media_link)
                        workLinkFile.write(str(working_links_count) + ". Link: " + link + "\n")
                        workLinkFile.write("Media Link: " + media_link + "\n")
                        workLinkFile.write("File Type: " + content_type + "\n\n")


        except Exception as e:
            print(f"{str(e)}")
```

Figure 33: Code for finding available medias from the formatted gcs output

The formatted links and found available media links are given below from the output files.



```
https://storage.googleapis.com/storage/v1/b/101/o
https://storage.googleapis.com/storage/v1/b/4ever/o
https://storage.googleapis.com/storage/v1/b/500/o
https://storage.googleapis.com/storage/v1/b/777777/o
https://storage.googleapis.com/storage/v1/b/a02/o
https://storage.googleapis.com/storage/v1/b/a12345678/o
https://storage.googleapis.com/storage/v1/b/a4a/o
https://storage.googleapis.com/storage/v1/b/a79/o
https://storage.googleapis.com/storage/v1/b/aa11/o
https://storage.googleapis.com/storage/v1/b/aa25/o
https://storage.googleapis.com/storage/v1/b/aadhaar/o
https://storage.googleapis.com/storage/v1/b/aagc/o
https://storage.googleapis.com/storage/v1/b/abbas/o
https://storage.googleapis.com/storage/v1/b/abcabc/o
https://storage.googleapis.com/storage/v1/b/abcd1234/o
https://storage.googleapis.com/storage/v1/b/ability/o
https://storage.googleapis.com/storage/v1/b/abonnement/o
https://storage.googleapis.com/storage/v1/b/about/o
https://storage.googleapis.com/storage/v1/b/academic/o
https://storage.googleapis.com/storage/v1/b/academia/o
https://storage.googleapis.com/storage/v1/b/accelerator/o
https://storage.googleapis.com/storage/v1/b/accent/o
https://storage.googleapis.com/storage/v1/b/accademia/o
https://storage.googleapis.com/storage/v1/b/accessibility/o
https://storage.googleapis.com/storage/v1/b/accnt/o
https://storage.googleapis.com/storage/v1/b/activity/o
https://storage.googleapis.com/storage/v1/b/ad1/o
https://storage.googleapis.com/storage/v1/b/adamant/o
https://storage.googleapis.com/storage/v1/b/addressbook/o
https://storage.googleapis.com/storage/v1/b/adforms/o
https://storage.googleapis.com/storage/v1/b/admin-dev/o
https://storage.googleapis.com/storage/v1/b/adminpanel/o
https://storage.googleapis.com/storage/v1/b/admin-staging/o
https://storage.googleapis.com/storage/v1/b/adops/o
https://storage.googleapis.com/storage/v1/b/adtest/o
https://storage.googleapis.com/storage/v1/b/adv1/o
https://storage.googleapis.com/storage/v1/b/advokat/o
```

Figure 34: Links after formatting the gcs output

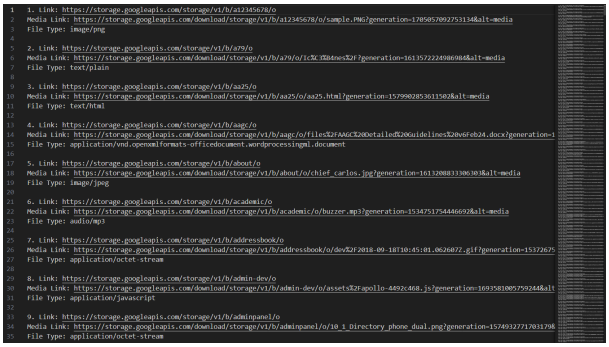


Figure 35: Available medias from the formatted gcs output

If we paste the available media links, we can find the available media files. The **output files**, python code and necessary files are given in the following github link: [2]

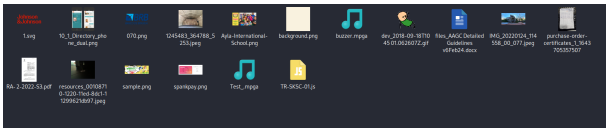


Figure 36: Available medias files from the gcs output

5.6 S3

In this mode, gobuster looks up for the words in the word files for amazon s3 buckets. Here, we have used a modified file, named **s3wordsv1.txt** The command for this mode is given below:

```
gobuster s3 -w s3wordsv1.txt -t 10 > s34.txt
```

Different parts of the command are explained below:

- s3: s3 mode of operation
- -w: Path to the wordlist file
- -t: number of threads
- : s34.txt: The output will be written to s34.txt file

The output for this command is given below:

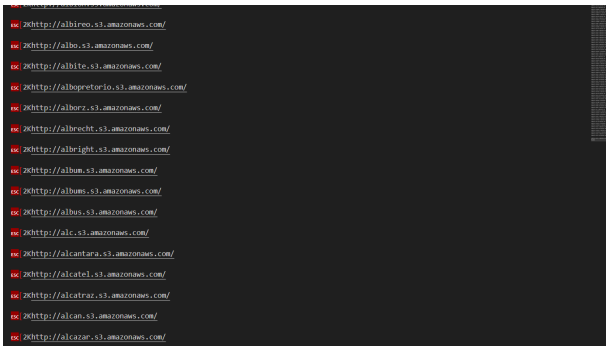


Figure 37: Output of the s3 mode

Most of the links are not accessible. They show the code **AccessDenied**.

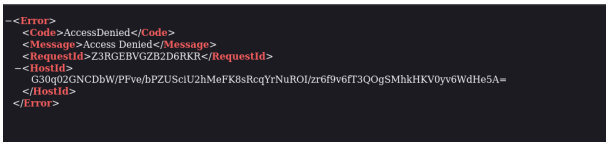


Figure 38: Output Access Denied the s3 mode

We found a link that gives us two links.

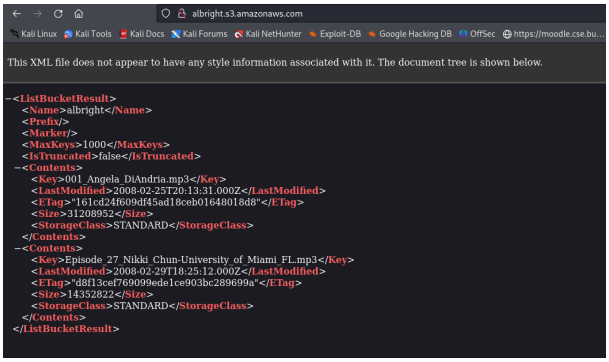


Figure 39: Output Success for the s3 mode

We can view the files by putting the file name after the link separated by a slash. Thus we can get publicly available amazon s3 bucket files.

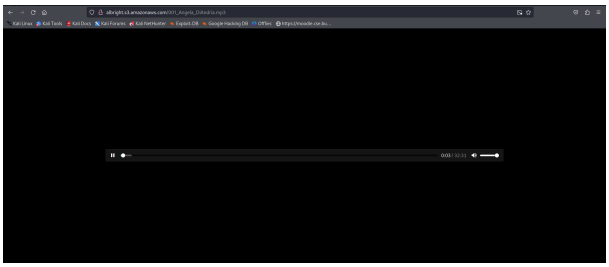


Figure 40: Output File for the s3 mode

The files that we found are given in the github link [2].

5.7 TFTP

TFTP (Trivial File Transfer Protocol) is a simple file transfer protocol. It does not have any authentication protocol like ftp servers. So, it is not suitable for remote networks. Rather, it is set up in a local environment. We set up a tftp server on localhost in **ubuntu**(as we faced some problems setting it up on kali linux), following this link [4]. The tftp server was set up on the directory **/tftpboot**. It had the following files:

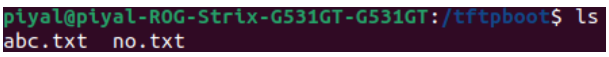


Figure 41: Files in the tftp server

Then we used gobuster to enumerate over the files and find any file mentioned in the **tftpwords.txt** file. The contents of the word file is given below:

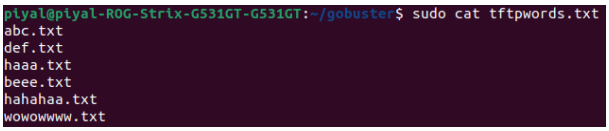


Figure 42: Word File for the tftp mode

The command for this operation is given below:

```
./gobuster tftp -s 127.0.0.1 -w tftpwords.txt
```

Different parts of the command are explained below:

- tftp: tftp mode of operation
- -w: Path to the wordlist file
- -s: Link to the tftp server

As we had to set up gobuster on ubuntu for this mode and ubuntu does not support the latest gobuster version, so we built an executable file to run gobuster from the github code. For this reason, we need to run the executable file each time using **./gobuster**. As the word file contains abc.txt, we found the abc.txt file. But no.txt file was not found as it was not in the word file. Thus, we can find files on our tftp server using gobuster.

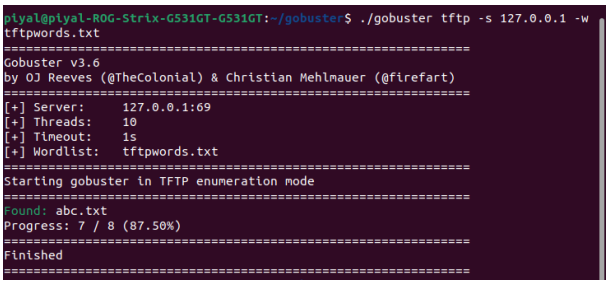


Figure 43: Output for the tftp mode

References

[1] *Github Repo For Gobuster*. URL: https://github.com/OJ/gobuster/.

[2] *Github Repo For Output Files and Modes*. URL: https://github.com/ferb97/CSE406-Security-Project/.

[3] *SecLists (Repo with wordlists we used)*. URL: https://github.com/danielmiessler/SecLists.

[4] *TFTP Setup*. URL: https://www.addictivetips.com/ubuntu-linux-tips/set-up-a-tftp-server-on-ubuntu-server/.