CSE 306 Project Report

GOBUSTER : A Content Discovery Tool

Report Prepared by:

Sheikh Hasanul Banna - 1805094
Partho Kunda - 1805107

Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology

Contents

1 Introduction

Gobuster is a content discovery tool that is used for discovering hidden directories, paths and files on a website. Gobuster has 7 basic modes of operation:

- **DIR :** The directory mode is used to brute force and find hidden directories, files etc. It is by far the most popular mode of gobuster.
- **DNS :** 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.
- **VHOST:** VHOST is the short form of Virtual Host. IN this mode we add hostname to the request header of our http requests and check the responses to our requests. This way we find out which websites are being hosted under the same IP address.
- **FUZZ :**Fuzzing is the process of finding different parameters on a websites. IN FUZZ mode, we can insert content from wordlists into different places of a websites and check the responses to understand which parameters are valid for that websites.
- **GCS :** IN this mode we look for publicly available google cloud storage buckets. This mode can be used to automate public documents extraction from google cloud storage.
- **S3 :** This works similarly to GCS. This mode is used to extract information from publicly available amazon S3 buckets.
- **TFTP :** TFTP is a more loose version of FTP, where we don't need any password to access the service and the server and client are both situated on the same LAN. TFTP mode of gobuster allows us to discover what files exist on a tftp server and then we can get the files with a very simple command.

A basic diagram of what gobuster does is given below:

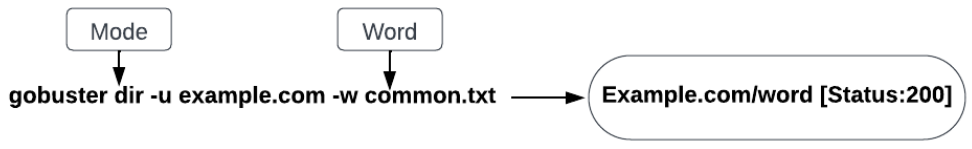

Mode
Word
gobuster dir -u example.com -w common.txt
Example.com/word [Status:200]

Figure 1: Basic diagram of Gobuster.

Basically we add a wordlist from which gobuster will take words and change the requests by merging the word with the requests. How the request will be modified will depend on our modes of operation. After modifying the request with words, the tool will process the request and record the responses.

2 Setting up Gobuster

- **Installation of Go :** Gobuster is written in Go. So you need to have go installed in your system. To check if its installed open terminal and type
  `go --version`
  If it's not installed, you need to install it from go binaries installations.
  From there, select the binary for your operating system and the binaries will start downloading.
  Once downloading is finished, you need to make sure any previous installation is removed. For this, run the following command on Linux:
  `rm -rf /usr/local/go && tar -C /usr/local -xzf go1.21.1.linux-amd64.tar.gz`
  Add the location of bin directory to path variable by going to /etc/profile and adding the following line:
  `export PATH=$PATH:/usr/local/go/bin`
  Check if it's installed successfully by running
  `go --version`
- **Installing Gobuster :** After the go environment is set up, just run
  `go install github.com/OJ/gobuster/v3@latest`
  Check if Gobuster Installation is successful, run the following command in terminal:
  `gobuster`

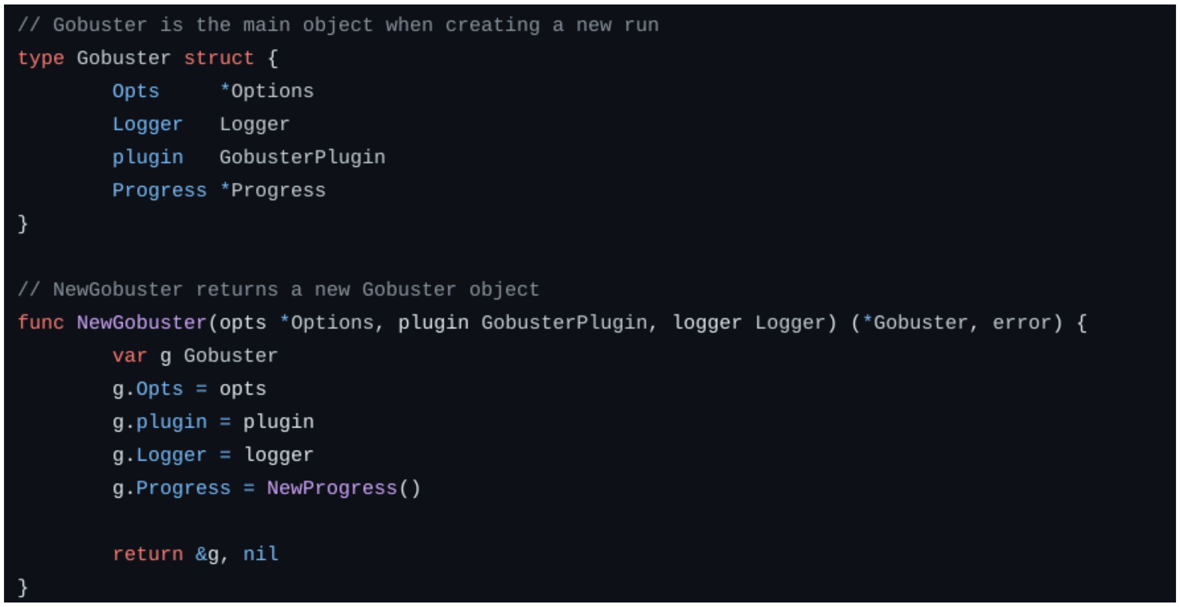
If all the steps were successful, you should be ready to use gobuster.

3 Source Code Overview and Short Description

Gobuster is written in go language, which is a lightweight and fast language. Go language is easy to use for concurrency, something that is important for running each of our commands in multi-thread mode. If we follow the github link of the project at https://github.com/OJ/gobuster,we can see that each mode of operation in gobuster is implemented in a different folder.

3.1 libgobuster

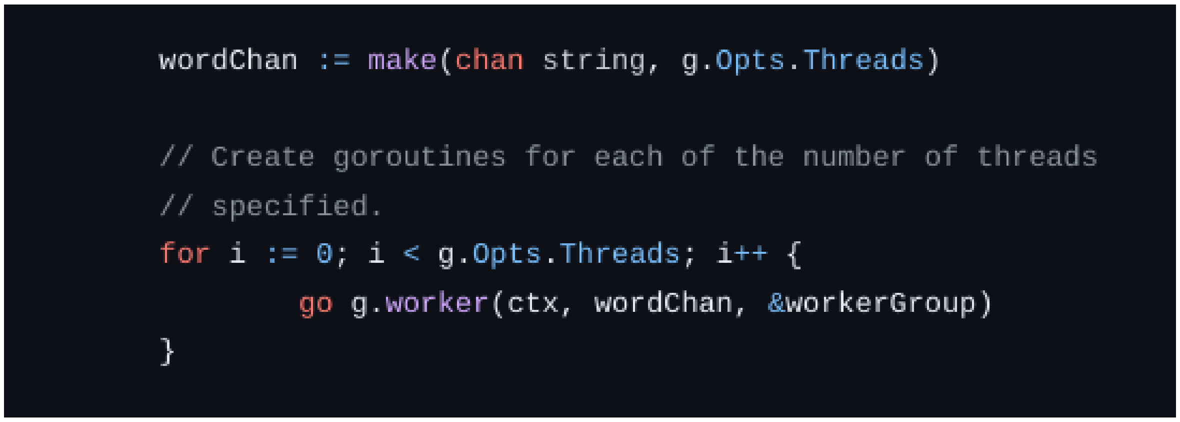
The libgobuster directory declares the basic interface of our gobuster object. going to /libgobuster/libgobuster.go, we can see the following code snippet:



```go
// Gobuster is the main object when creating a new run
type Gobuster struct {
        Opts     *Options
        Logger   Logger
        plugin   GobusterPlugin
        Progress *Progress
}

// NewGobuster returns a new Gobuster object
func NewGobuster(opts *Options, plugin GobusterPlugin, logger Logger) (*Gobuster, error) {
        var g Gobuster
        g.Opts = opts
        g.plugin = plugin
        g.Logger = logger
        g.Progress = NewProgress()

        return &g, nil
}
```

Figure 2: Basic interface of gobuster object.

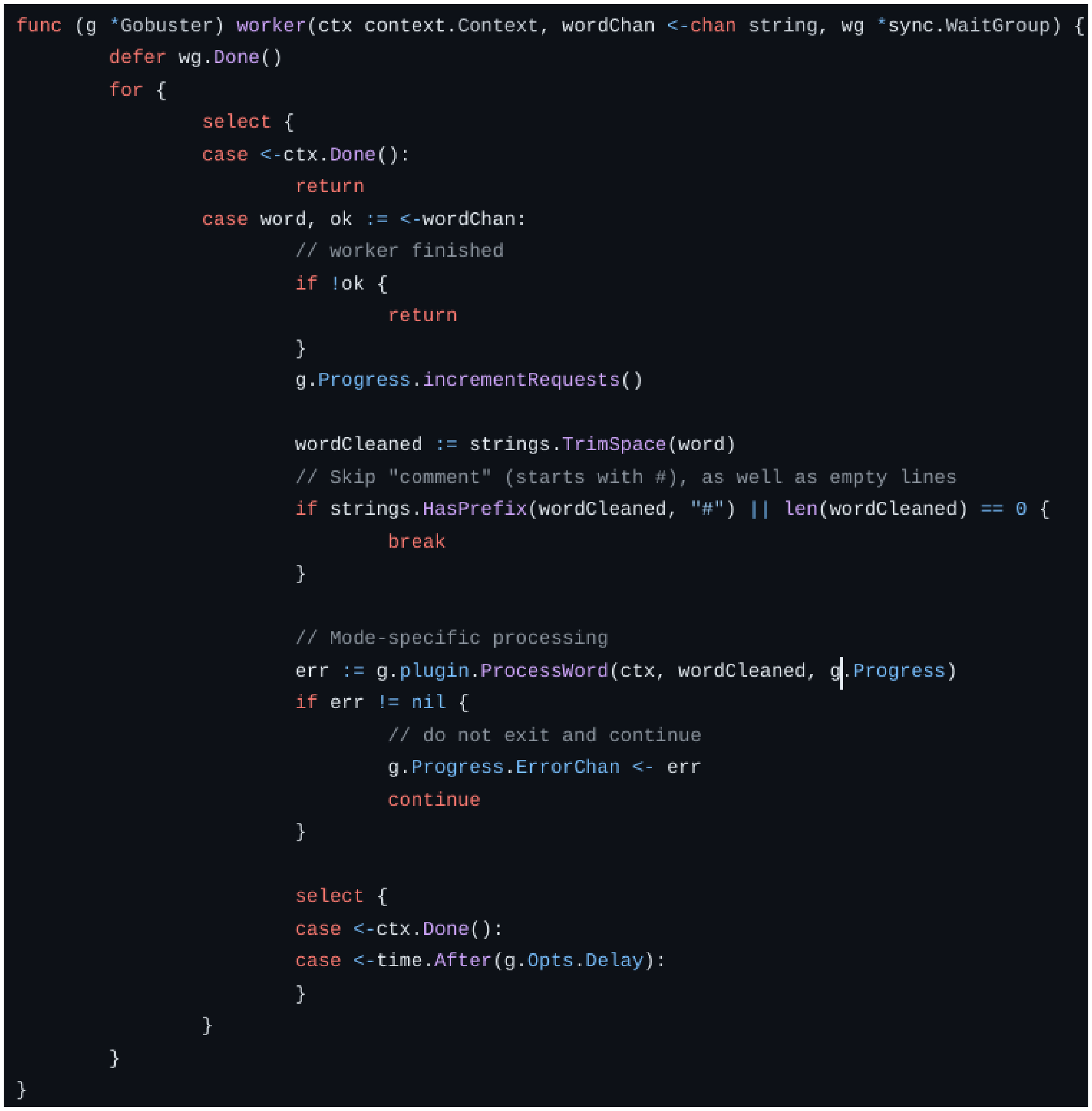
opts will define our mode of operation and related flags of that mode. Each of our modes of operation extends this gobuster struct to use it as their own property. In run function, we see the following snippet:



```go
        wordChan := make(chan string, g.Opts.Threads)

        // Create goroutines for each of the number of threads
        // specified.
        for i := 0; i < g.Opts.Threads; i++ {
                go g.worker(ctx, wordChan, &workerGroup)
        }
```

Figure 3: run function of libgobuster.

This shows us that a wordchannel is created for taking words for each iteration and then passing them to a function named worker. The following snippet shows the worker function:



```go
func (g *Gobuster) worker(ctx context.Context, wordChan <-chan string, wg *sync.WaitGroup) {
	defer wg.Done()
	for {
		select {
		case <-ctx.Done():
			return
		case word, ok := <-wordChan:
			// worker finished
			if !ok {
				return
			}
			g.Progress.incrementRequests()

			wordCleaned := strings.TrimSpace(word)
			// Skip "comment" (starts with #), as well as empty lines
			if strings.HasPrefix(wordCleaned, "#") || len(wordCleaned) == 0 {
				break
			}

			// Mode-specific processing
			err := g.plugin.ProcessWord(ctx, wordCleaned, g.Progress)
			if err != nil {
				// do not exit and continue
				g.Progress.ErrorChan <- err
				continue
			}

			select {
			case <-ctx.Done():
			case <-time.After(g.Opts.Delay):
			}
		}
	}
}
```

Figure 4: worker function of libgobuster.

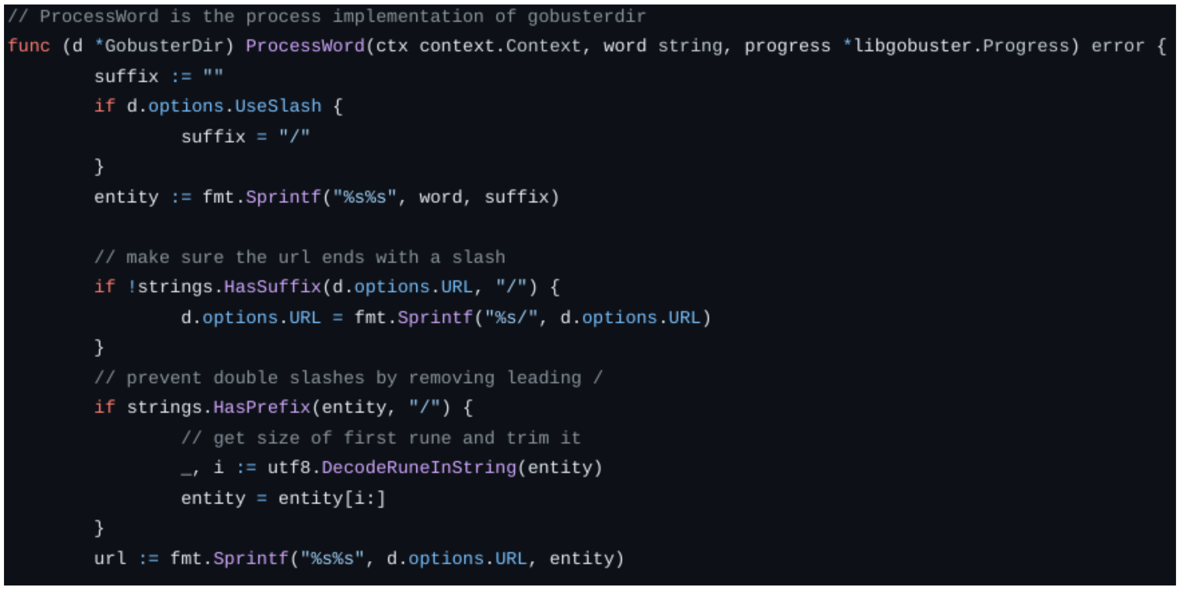
The call to `ProcessWord` function does all the mode specific functionalities. Now we are ready to view all the individual modes of operation.

3.2 gobusterdir

This mode finds the directories and files in a website by taking words from wordlist and then creating http request to a website `example.com/word`. The basic command of dir mode is as follows:

```
gobuster dir -u https://example.com -c -w common-files.txt
```

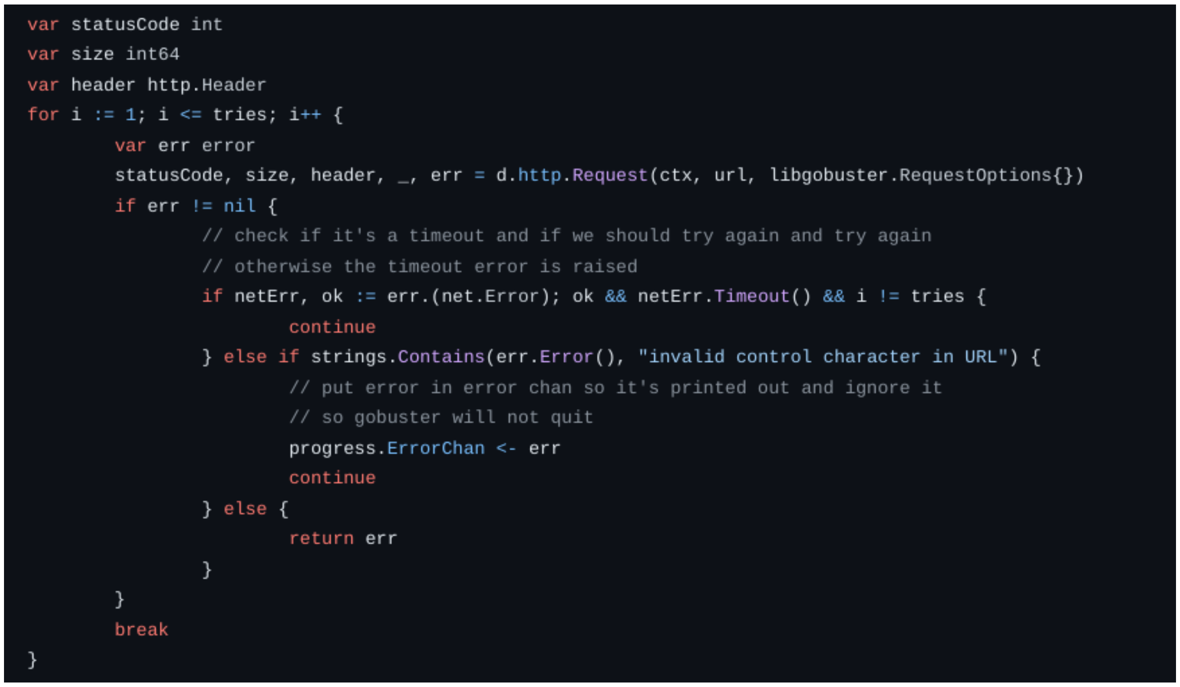
If we follow the /gobusterdir/gobusterdir.go file we can see the following code snippet:



```go
// ProcessWord is the process implementation of gobusterdir
func (d *GobusterDir) ProcessWord(ctx context.Context, word string, progress *libgobuster.Progress) error {
	suffix := ""
	if d.options.UseSlash {
		suffix = "/"
	}
	entity := fmt.Sprintf("%s%s", word, suffix)

	// make sure the url ends with a slash
	if !strings.HasSuffix(d.options.URL, "/") {
		d.options.URL = fmt.Sprintf("%s/", d.options.URL)
	}
	// prevent double slashes by removing leading /
	if strings.HasPrefix(entity, "/") {
		// get size of first rune and trim it
		_, i := utf8.DecodeRuneInString(entity)
		entity = entity[i:]
	}
	url := fmt.Sprintf("%s%s", d.options.URL, entity)
```

Figure 5: ProcessWord of gobusterdir

urls are generated by taking words from wordChan, handling "/" and then adding the word to `d.options.URL`. this creates urls like `example.com/word`.



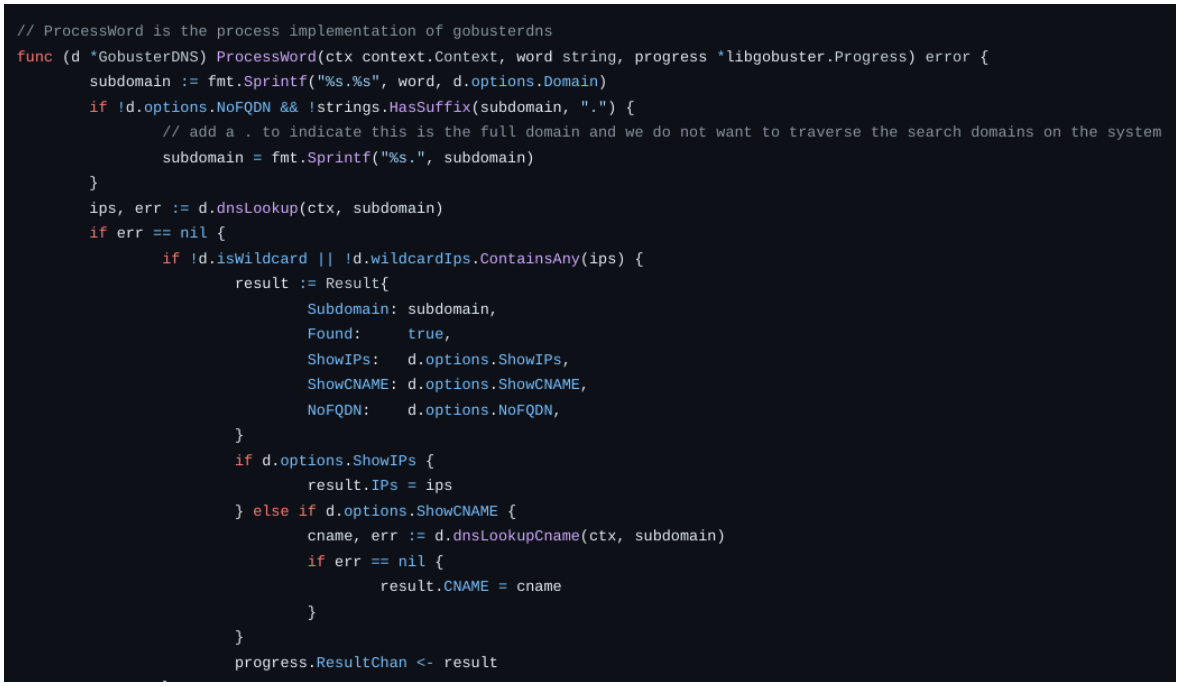
```go
var statusCode int
var size int64
var header http.Header
for i := 1; i <= tries; i++ {
	var err error
	statusCode, size, header, _, err = d.http.Request(ctx, url, libgobuster.RequestOptions{})
	if err != nil {
		// check if it's a timeout and if we should try again and try again
		// otherwise the timeout error is raised
		if netErr, ok := err.(net.Error); ok && netErr.Timeout() && i != tries {
			continue
		} else if strings.Contains(err.Error(), "invalid control character in URL") {
			// put error in error chan so it's printed out and ignore it
			// so gobuster will not quit
			progress.ErrorChan <- err
			continue
		} else {
			return err
		}
	}
	break
}
```

Figure 6: ProcessWord of gobusterdir contd.

In the above snippet, we send a http request using `d.http.Request`. We get statusCode, size of response from the return value, which are enough for our basic dir operations.

3.3 gobusterdns

In this mode, we add words from the wordlist and create subdomains like `word.example.com` then send dnsquery to the subdomain. The code snippet is given below:



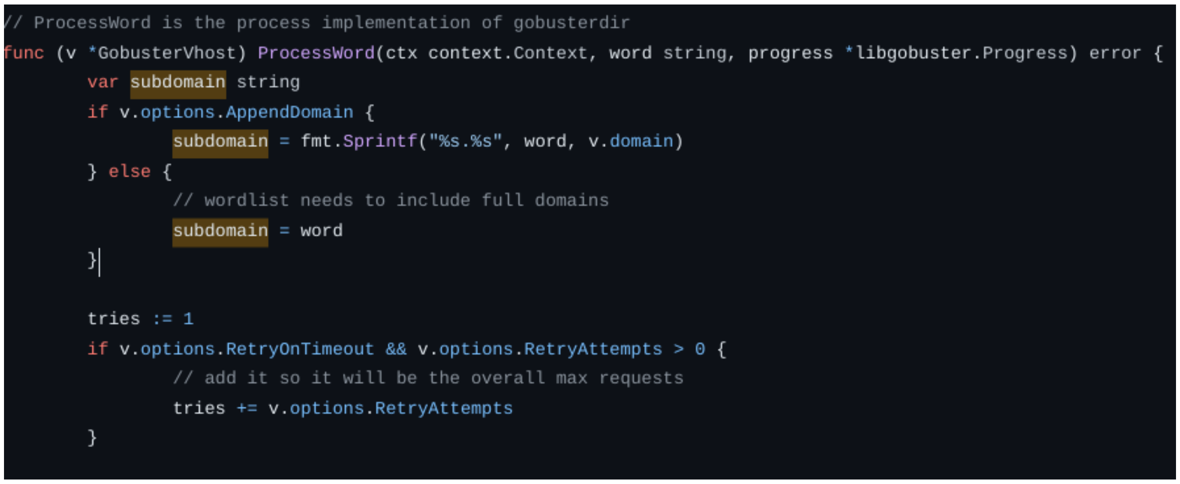
```go
// ProcessWord is the process implementation of gobusterdns
func (d *GobusterDNS) ProcessWord(ctx context.Context, word string, progress *libgobuster.Progress) error {
        subdomain := fmt.Sprintf("%s.%s", word, d.options.Domain)
        if !d.options.NoFQDN && !strings.HasSuffix(subdomain, ".") {
                // add a . to indicate this is the full domain and we do not want to traverse the search domains on the system
                subdomain = fmt.Sprintf("%s.", subdomain)
        }
        ips, err := d.dnsLookup(ctx, subdomain)
        if err == nil {
                if !d.isWildcard || !d.wildcardIps.ContainsAny(ips) {
                        result := Result{
                                Subdomain: subdomain,
                                Found:     true,
                                ShowIPs:   d.options.ShowIPs,
                                ShowCNAME: d.options.ShowCNAME,
                                NoFQDN:    d.options.NoFQDN,
                        }
                        if d.options.ShowIPs {
                                result.IPs = ips
                        } else if d.options.ShowCNAME {
                                cname, err := d.dnsLookupCname(ctx, subdomain)
                                if err == nil {
                                        result.CNAME = cname
                                }
                        }
                        progress.ResultChan <- result
```

Figure 7: ProcessWord of gobusterdns

subdomain is created by appending `word` to `d.options.Domain`
The line [`ips, err := d.dnsLookup(ctx, subdomain)`]sends dns query to dns server. the result is saved in the object. We can extract or filter from the result afterwards.

3.4 gobustervhost

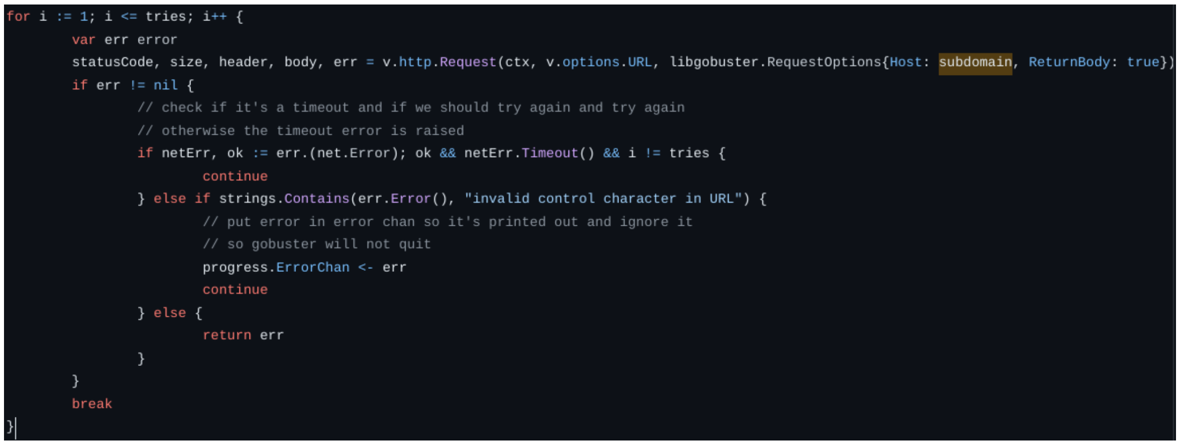
For looking for sites that are hosted virtually by the same website, we send http requests to same website with different hostname in header. we first create the subdomain name as follows:



```go
// ProcessWord is the process implementation of gobusterdir
func (v *GobusterVhost) ProcessWord(ctx context.Context, word string, progress *libgobuster.Progress) error {
	var subdomain string
	if v.options.AppendDomain {
		subdomain = fmt.Sprintf("%s.%s", word, v.domain)
	} else {
		// wordlist needs to include full domains
		subdomain = word
	}

	tries := 1
	if v.options.RetryOnTimeout && v.options.RetryAttempts > 0 {
		// add it so it will be the overall max requests
		tries += v.options.RetryAttempts
	}
```

Figure 8: ProcessWord of gobustervhost

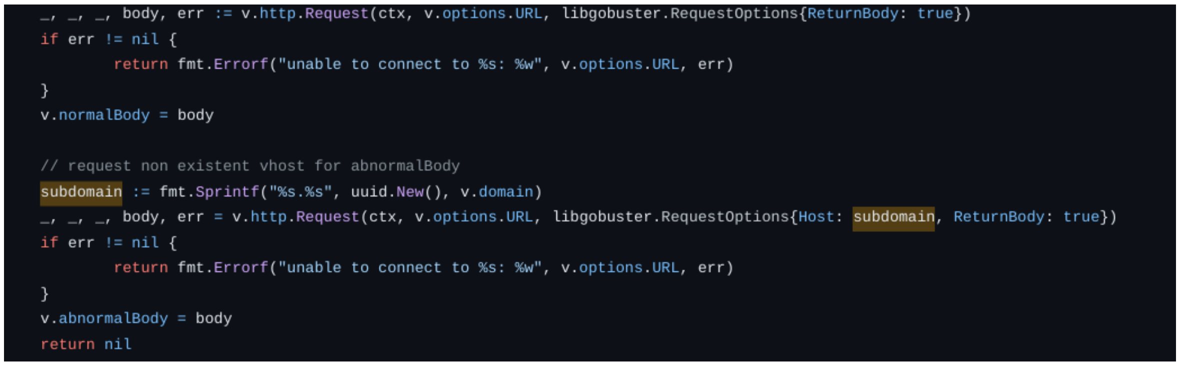
After creating subdomain string, we send htttp Request as follows:



```go
for i := 1; i <= tries; i++ {
	var err error
	statusCode, size, header, body, err = v.http.Request(ctx, v.options.URL, libgobuster.RequestOptions{Host: subdomain, ReturnBody: true})
	if err != nil {
		// check if it's a timeout and if we should try again and try again
		// otherwise the timeout error is raised
		if netErr, ok := err.(net.Error); ok && netErr.Timeout() && i != tries {
			continue
		} else if strings.Contains(err.Error(), "invalid control character in URL") {
			// put error in error chan so it's printed out and ignore it
			// so gobuster will not quit
			progress.ErrorChan <- err
			continue
		} else {
			return err
		}
	}
	break
}
```

Figure 9: Sending http request for vhost

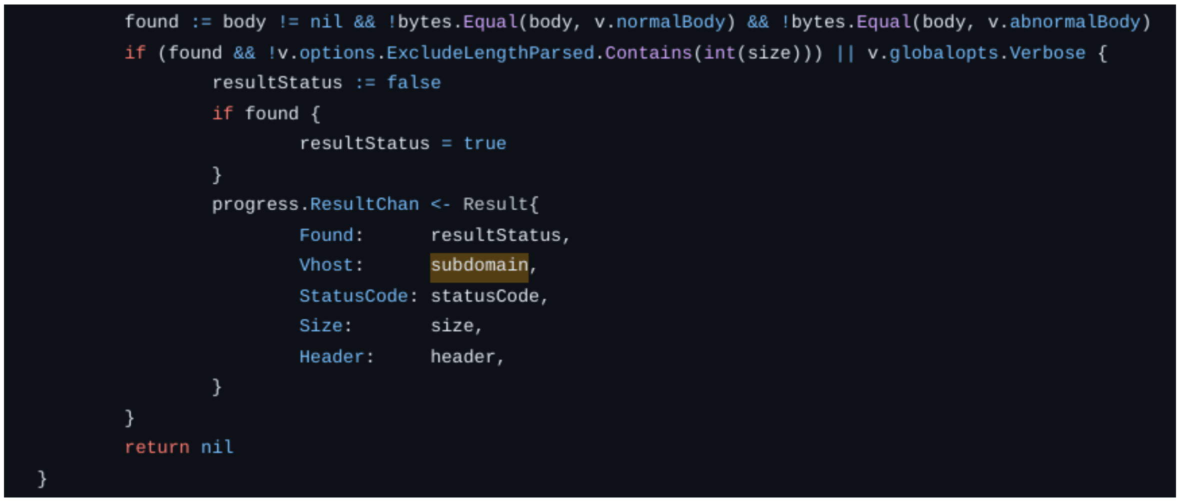
now we have status code, response size,body etc. of the response. Next our job is to see if the response body is the same as default response body or error response body. we send a http request to the domain name normally to get normal body. we also send abnormal request using a uuid generator and save the response body as abnormal body. The following figure shows this:



```go
_, _, _, body, err := v.http.Request(ctx, v.options.URL, libgobuster.RequestOptions{ReturnBody: true})
if err != nil {
        return fmt.Errorf("unable to connect to %s: %w", v.options.URL, err)
}
v.normalBody = body

// request non existent vhost for abnormalBody
subdomain := fmt.Sprintf("%s.%s", uuid.New(), v.domain)
_, _, _, body, err = v.http.Request(ctx, v.options.URL, libgobuster.RequestOptions{Host: subdomain, ReturnBody: true})
if err != nil {
        return fmt.Errorf("unable to connect to %s: %w", v.options.URL, err)
}
v.abnormalBody = body
return nil
```

Figure 10: getting normal and abnormal body for vhost

And 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.



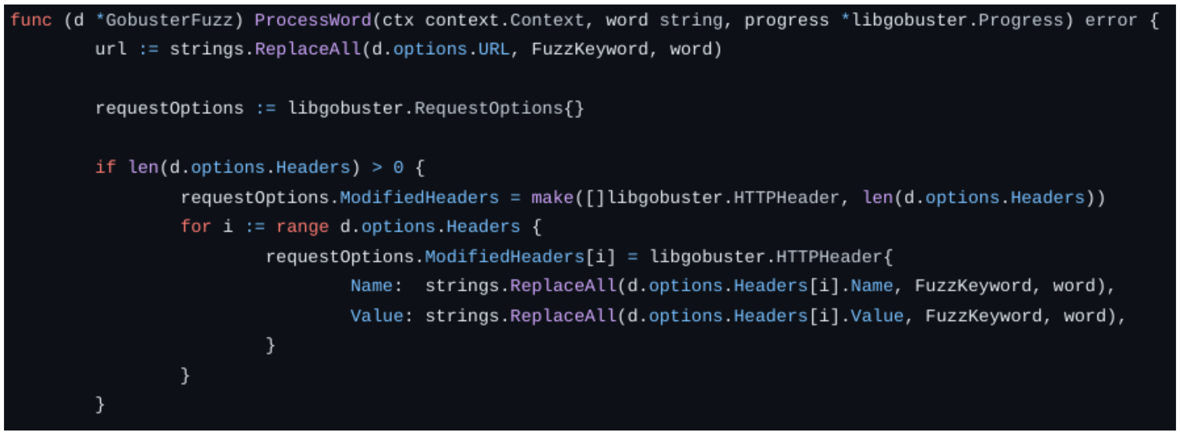
```go
        found := body != nil && !bytes.Equal(body, v.normalBody) && !bytes.Equal(body, v.abnormalBody)
        if (found && !v.options.ExcludeLengthParsed.Contains(int(size))) || v.globalopts.Verbose {
                resultStatus := false
                if found {
                        resultStatus = true
                }
                progress.ResultChan <- Result{
                        Found:      resultStatus,
                        Vhost:      subdomain,
                        StatusCode: statusCode,
                        Size:       size,
                        Header:     header,
                }
        }
        return nil
}
```

Figure 11: comparing response body for gobustervhost

If the conditions are fulfilled, we get a new website hosted by the base domain, we add that to our results. This we get a list of virtually hosted websites.

3.5 gobusterfuzz

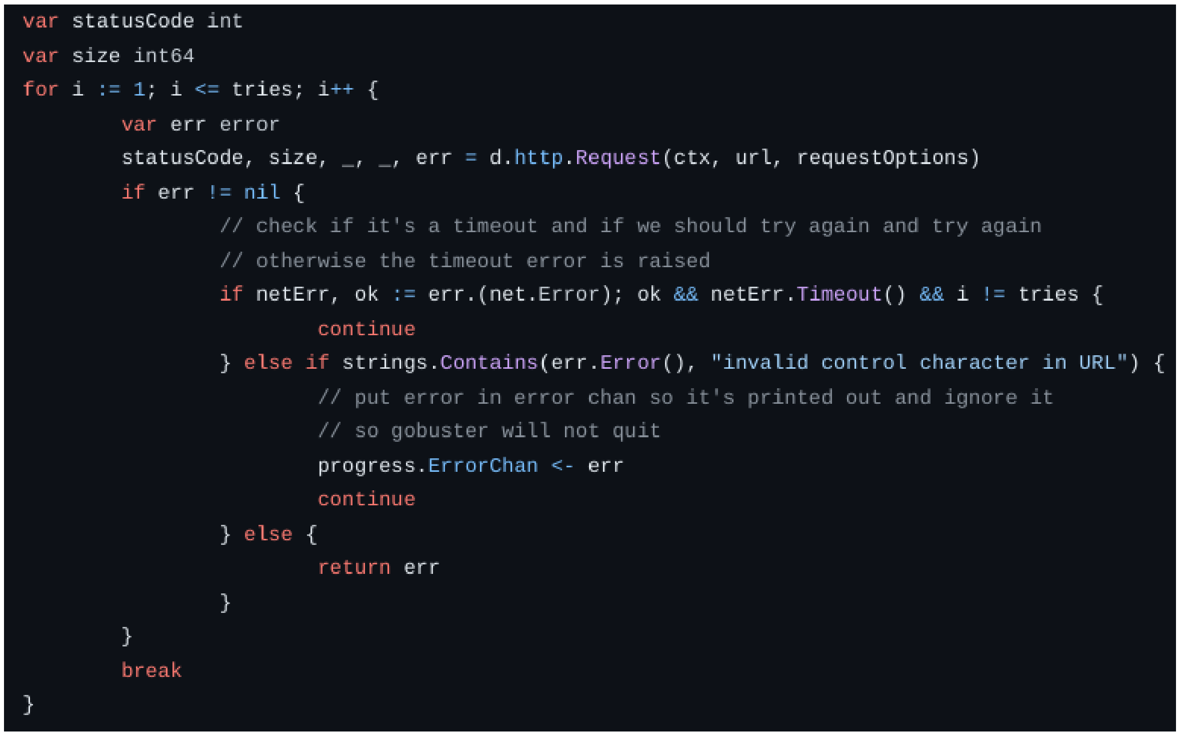
In this mode we modify the provided url by placing words from our wordlist at different places of the url and seeing the responses.



```go
func (d *GobusterFuzz) ProcessWord(ctx context.Context, word string, progress *libgobuster.Progress) error {
	url := strings.ReplaceAll(d.options.URL, FuzzKeyword, word)

	requestOptions := libgobuster.RequestOptions{}

	if len(d.options.Headers) > 0 {
		requestOptions.ModifiedHeaders = make([]libgobuster.HTTPHeader, len(d.options.Headers))
		for i := range d.options.Headers {
			requestOptions.ModifiedHeaders[i] = libgobuster.HTTPHeader{
				Name:  strings.ReplaceAll(d.options.Headers[i].Name, FuzzKeyword, word),
				Value: strings.ReplaceAll(d.options.Headers[i].Value, FuzzKeyword, word),
			}
		}
	}
```

Figure 12: creating url and request body for fuzzing

basic command for fuzz looks like this:
`gobuster fuzz -u https://example.com?FUZZ=test -w names.txt`
Here we are replacing the keyword FUZZ in url with words from names.txt. The snippet above shows this process. Then we send the request to modified url and record the responses.



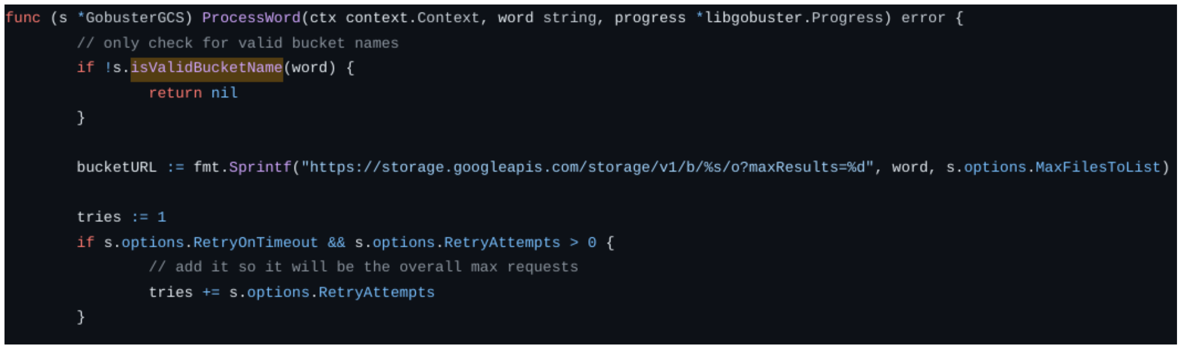
```go
var statusCode int
var size int64
for i := 1; i <= tries; i++ {
	var err error
	statusCode, size, _, _, err = d.http.Request(ctx, url, requestOptions)
	if err != nil {
		// check if it's a timeout and if we should try again and try again
		// otherwise the timeout error is raised
		if netErr, ok := err.(net.Error); ok && netErr.Timeout() && i != tries {
			continue
		} else if strings.Contains(err.Error(), "invalid control character in URL") {
			// put error in error chan so it's printed out and ignore it
			// so gobuster will not quit
			progress.ErrorChan <- err
			continue
		} else {
			return err
		}
	}
	break
}
```

Figure 13: sending http request for fuzzing

We examine the recorded status code and size to determine which of the requests were actually accepted by the website. This way we can fuzz for different properties of the webiste.

3.6 gobustergcs

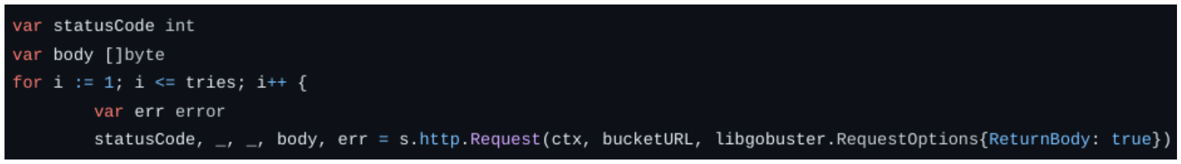
This mode is used to find publicly available gcs buckets. By finding these buckets, we can even access some of their contents by accessing the links provided in the response. The basic command for gcs mode looks like this:
`gobuster gcs -w bucket-names.txt`
for gcs buckets, the url are basically in the format :
`https://storage.googleapis.com/storage/v1/b/bucket/o`
So the main operation in this mode is to brute force the link with bucket names from wordlist and viewing the responses. The code snippet below shows this:



```
func (s *GobusterGCS) ProcessWord(ctx context.Context, word string, progress *libgobuster.Progress) error {
        // only check for valid bucket names
        if !s.isValidBucketName(word) {
                return nil
        }

        bucketURL := fmt.Sprintf("https://storage.googleapis.com/storage/v1/b/%s/o?maxResults=%d", word, s.options.MaxFilesToList)

        tries := 1
        if s.options.RetryOnTimeout && s.options.RetryAttempts > 0 {
                // add it so it will be the overall max requests
                tries += s.options.RetryAttempts
        }
```

Figure 14: Creating url for buckets in gcs

After creating the url, we send http requests to them. Gobuster gives us the status code of the responses and we investigate them to see which bucket names are actual public buckets. If we find the bucket names, we can simply visit the links again to view what type of contents are in the bucket.



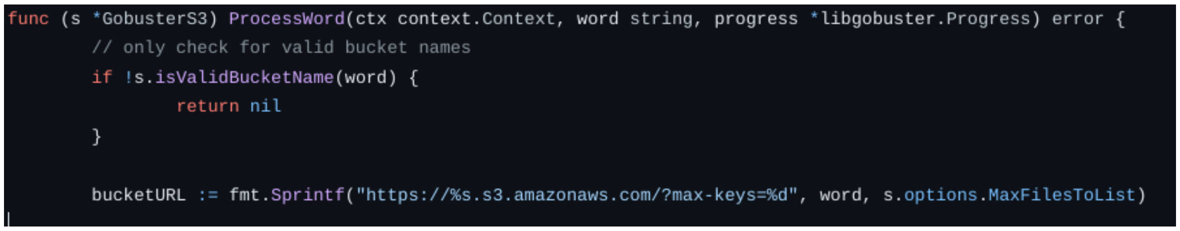
```
var statusCode int
var body []byte
for i := 1; i <= tries; i++ {
        var err error
        statusCode, _, _, body, err = s.http.Request(ctx, bucketURL, libgobuster.RequestOptions{ReturnBody: true})
```

Figure 15: request and response for gcs

3.7 gobusters3

This mode works almost identically to gcs. The difference is in the url that is required to access public S3 bucket of amazon. The url for s3 buckets is:
`https://bucket.s3.amazonaws.com/`

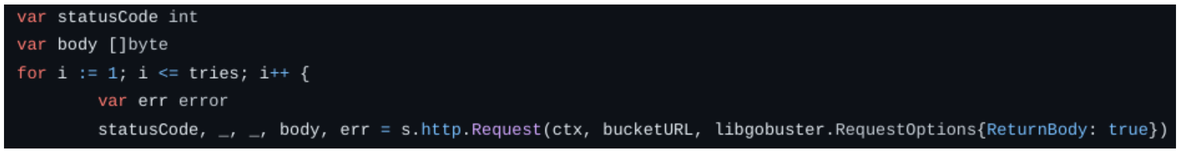
the basic command for running s3 mode is:
`gobuster s3 -w bucket-names.txt`
url generation for gobuster s3:



```
func (s *GobusterS3) ProcessWord(ctx context.Context, word string, progress *libgobuster.Progress) error {
	// only check for valid bucket names
	if !s.isValidBucketName(word) {
		return nil
	}

	bucketURL := fmt.Sprintf("https://%s.s3.amazonaws.com/?max-keys=%d", word, s.options.MaxFilesToList)
```

Figure 16: url generation for S3

Process of sending request and recording response is shown below:

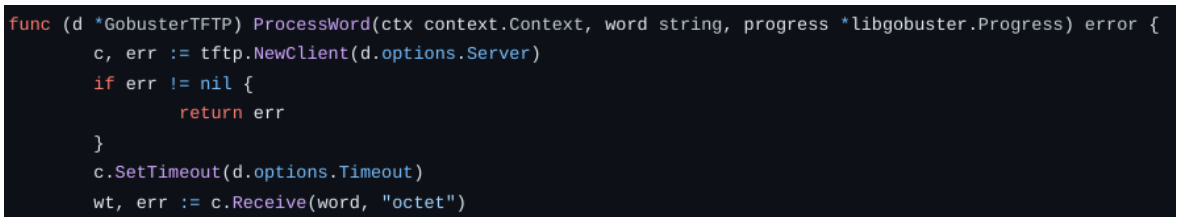


```
var statusCode int
var body []byte
for i := 1; i <= tries; i++ {
	var err error
	statusCode, _, _, body, err = s.http.Request(ctx, bucketURL, libgobuster.RequestOptions{ReturnBody: true})
```

Figure 17: request and response of s3

3.8 gobustetftp

As discussed in introduction section, tftp server doesn't require any password to access them. However, we can't view the contents of tftp server from client's end directly. So we use brute forcing of gobuster to find some filenames in a particular tftp server. The basic command for tftp mode is as follows:
`gobuster tftp -s tftp.example.com -w common-filenames.txt`
The code snippet for tftp is given below:



```
func (d *GobusterTFTP) ProcessWord(ctx context.Context, word string, progress *libgobuster.Progress) error {
	c, err := tftp.NewClient(d.options.Server)
	if err != nil {
		return err
	}
	c.SetTimeout(d.options.Timeout)
	wt, err := c.Receive(word, "octet")
```

Figure 18: connecting and accessing file of s3

Here we are connecting client to the given server address. Then we are enumerating our wordlist for words and sending a request to receive files having the same name as our word. As tftp doesn't have any protection, the file will be received by us and err will be nil(empty). Thus we will find the filename in tftp server.

4 Documentation For The Features

If the installation guide provided in section 2 was followed thoroughly, running the features is very straightforward. We simply need to run simple commands. For each features we have different commands and flags. To view basic features and flags:

```
gobuster -h
```

4.1 DIR

The main command structure for dir is as follows:

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

- dir : this indicates we are using dir mode
- -u : this indicates that the string following this flag is an url which will be used for dir attack.
- -w : wordlist flag. The string following this will be the filename which contains words we will use to brute force the url.

the flags of dir are as follows:

- -c, –cookies string : Cookies to use for the requests
- –exclude-length : exclude the following content length.
- -e, –expanded : Expanded mode, print full URLs
- -x, –extensions string : File extension(s) to search for
- -r, –follow-redirect : Follow redirects
- -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")
- -n, –no-status : Don't print status codes
- -P, –password string : Password for Basic Auth
- –proxy string : Proxy to use for requests [http(s)://host:port]
- -s, –status-codes string : Positive status codes(accept)
- -b, –status-blacklist : Negative status codes (will override status-codes if set) (default "404")

- –timeout duration : HTTP Timeout (default 10s)
- -u, –url string : The target URL
- -a, –useragent string : Set the User-Agent string (default "gobuster/3.2.0")
- -U, –username string : Username for Basic Auth
- -t, –threads int : Number of concurrent threads (default 10)
- -w, –wordlist string : Path to the wordlist

4.2 DNS

The main command structure for dir is as follows:
`gobuster dns -d example.com -t 50 -w common-names.txt`

- dns : indicates that we are operating in dns mode.
- -d : the domain name string will be followed by this flag
- -t : thread count concurrent number of threads that will serve the request in parallel.
-

the flags of dns are as follows:

- -d, –domain string : The target domain
- -h, –help :help for dns
- -c, –show-cname : Show CNAME records (cannot be used with '-i' option)
- -i, –show-ips : Show IP addresses
- –wildcard : Force continued operation when wildcard found
- -t, –threads int : Number of concurrent threads (default 10)
- -w, –wordlist string : Path to the wordlist

4.3 VHOST

The main command structure for vhost is as follows:

```
gobuster vhost -u https://example.com -w common-vhosts.txt --append-domain
```

- vhost : indicates that we are in vhost mode
- -u : url string of base web site
- -w : wordlist that contain vhost names
- –append-domain : add domain name to the words. for example,word.google.com

4.4 FUZZ

The main command structure for FUZZ is as follows:

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

- fuzz : indicates that we are in fuzz mode
- FUZZ : the wordlist's contents will be replaced by this keyword to make a valid request.

4.5 gcs

The main command structure for gcs is as follows:

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

- gcs : indicates we are in gcs mode.
- -w : wordlist file for buckets. we will look for responses of gcs request(only public buckets will respond positively)

4.6 s3

The main command structure for s3 is as follows:

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

- s3 : indicates we are in s3 mode.
- -w : wordlist file of buckets. we will look for responses of s3 request(only public buckets will respond positively)

4.7 tftp

The main command structure for tftp is as follows:

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

- tftp : indicates that we are in tftp mode
- -s : server string (address of the tftp server)
- -w: wordlist for potential filenames in tftp server.

5 Demonstration of the features

5.1 DIR

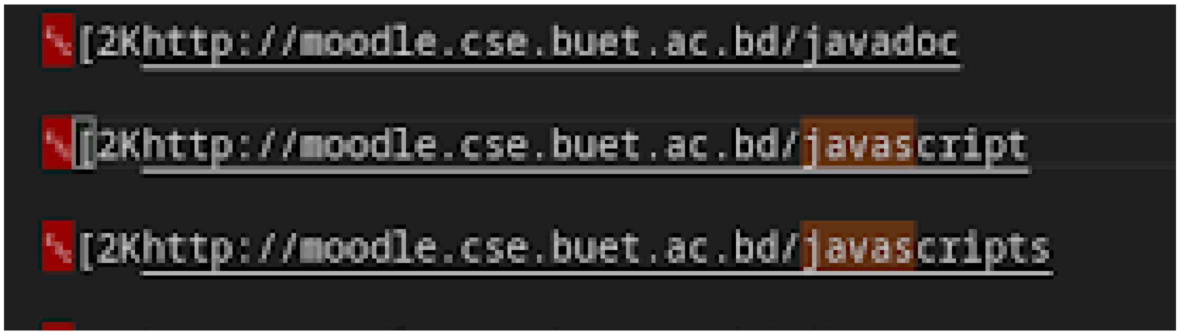
We will try to access a hidden file in moodle.cse.buet.ac.bd. We will run the following command :

```
gobuster dir -u moodle.cse.buet.ac.bd -w /common.txt -t 20 -e --exclude-length
361 -r > dirtest1.txt
```

Explanation of command:

- -u : we set url to moodle.cse.buet.ac.bd
- -w : we select common.txt as our wordlist of directories.
- -t : 20 threads will be used to run the command
- -e : this flag was used to print the whole directory
- –exclude-length : nonexisting urls returned size 361, we ignored them.
- -r : if a link redirected to another, we followed it
- dirtest1.txt : finally we saved the output to a file

dirtest1.txt contained some directory names, we selected the following among them:



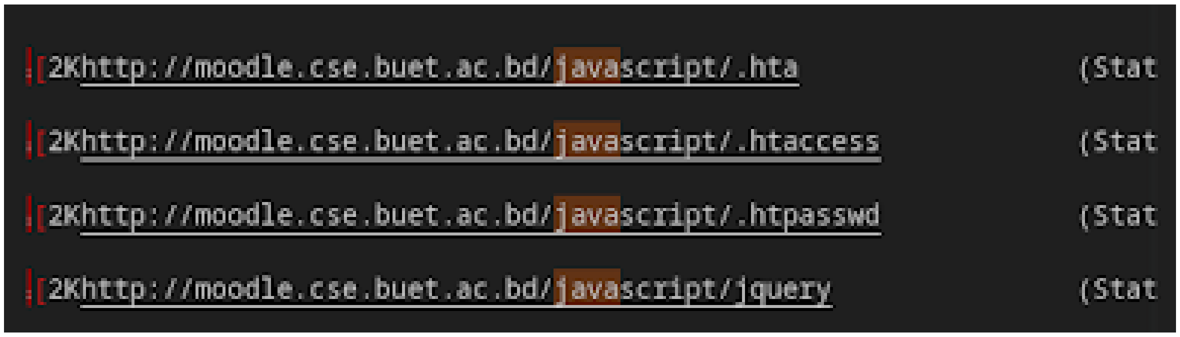
```
[2Khttp://moodle.cse.buet.ac.bd/javadoc
[2Khttp://moodle.cse.buet.ac.bd/javascript
[2Khttp://moodle.cse.buet.ac.bd/javascripts
```

Figure 19: javascript directory in moodle

The javascript directory will be used as dir start point next.

```
gobuster dir -u moodle.cse.buet.ac.bd/javascript -w /common.txt -t 20 -e
--exclude-length 361 -r > dirtest2.txt
```

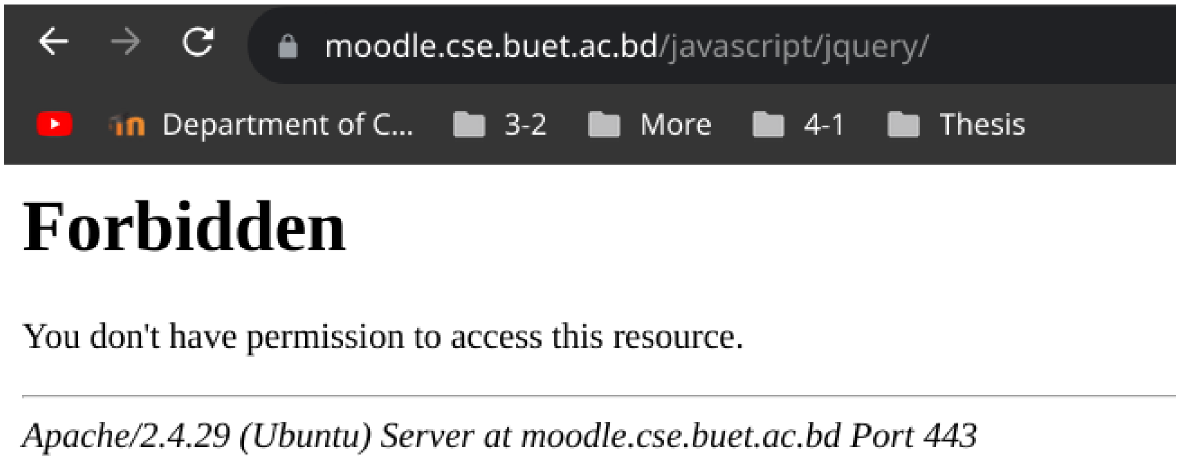
The output of this will be saved in dirtest2.txt



```
[2Khttp://moodle.cse.buet.ac.bd/javascript/.hta                    (Stat
[2Khttp://moodle.cse.buet.ac.bd/javascript/.htaccess               (Stat
[2Khttp://moodle.cse.buet.ac.bd/javascript/.htpasswd               (Stat
[2Khttp://moodle.cse.buet.ac.bd/javascript/jquery                  (Stat
```

Figure 20: jquery in javascript directory

If we try to access jquery, we will get forbidden error. The output of this will be saved in dirtest2.txt



Forbidden

You don't have permission to access this resource.

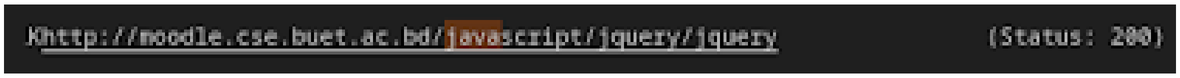
Apache/2.4.29 (Ubuntu) Server at moodle.cse.buet.ac.bd Port 443

Figure 21: jquery forbidden in javascript directory

And finally, we run the same command on javascript/jquery.

```
gobuster dir -u moodle.cse.buet.ac.bd/javascript/jquery -w /common.txt -t
20 -e --exclude-length 361 -r > dirtest2.txt
```

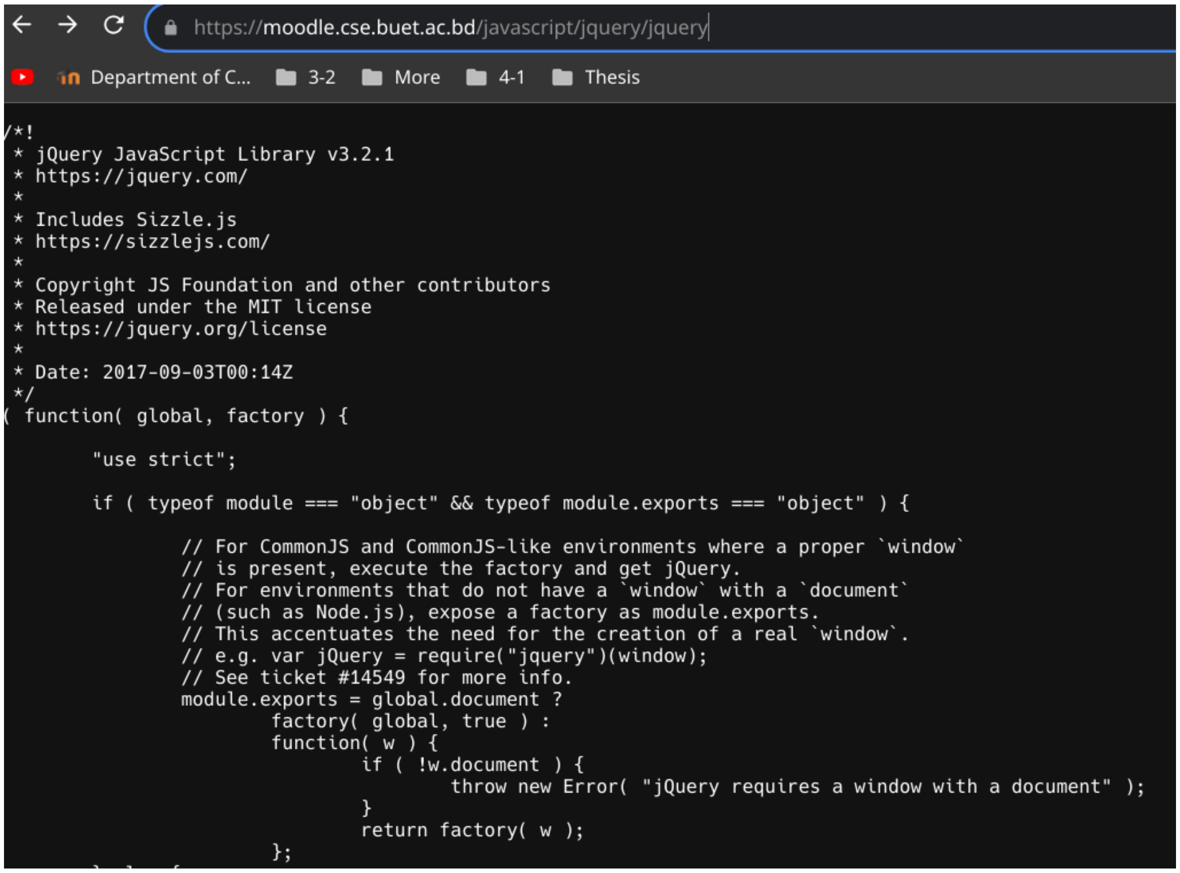
content of the file:



```
Khttp://moodle.cse.buet.ac.bd/javascript/jquery/jquery                    (Status: 200)
```

Figure 22: output of dir attack on javascript/jquery

We can access the file by going to moodle.cse.buet.ac.bd/javascript/jquery/jquery



```
https://moodle.cse.buet.ac.bd/javascript/jquery/jquery
```

```
/*!
 * jQuery JavaScript Library v3.2.1
 * https://jquery.com/
 *
 * Includes Sizzle.js
 * https://sizzlejs.com/
 *
 * Copyright JS Foundation and other contributors
 * Released under the MIT license
 * https://jquery.org/license
 *
 * Date: 2017-09-03T00:14Z
 */
( function( global, factory ) {

	"use strict";

	if ( typeof module === "object" && typeof module.exports === "object" ) {

		// For CommonJS and CommonJS-like environments where a proper `window`
		// is present, execute the factory and get jQuery.
		// For environments that do not have a `window` with a `document`
		// (such as Node.js), expose a factory as module.exports.
		// This accentuates the need for the creation of a real `window`.
		// e.g. var jQuery = require("jquery")(window);
		// See ticket #14549 for more info.
		module.exports = global.document ?
			factory( global, true ) :
			function( w ) {
				if ( !w.document ) {
					throw new Error( "jQuery requires a window with a document" );
				}
				return factory( w );
			};
```

Figure 23: browser view of javascript/jquery/jquery

So we can access hidden files and directories using the dir mode.

5.2 DNS

We will try to find some subdomains of google.com and their IP address through this mode.

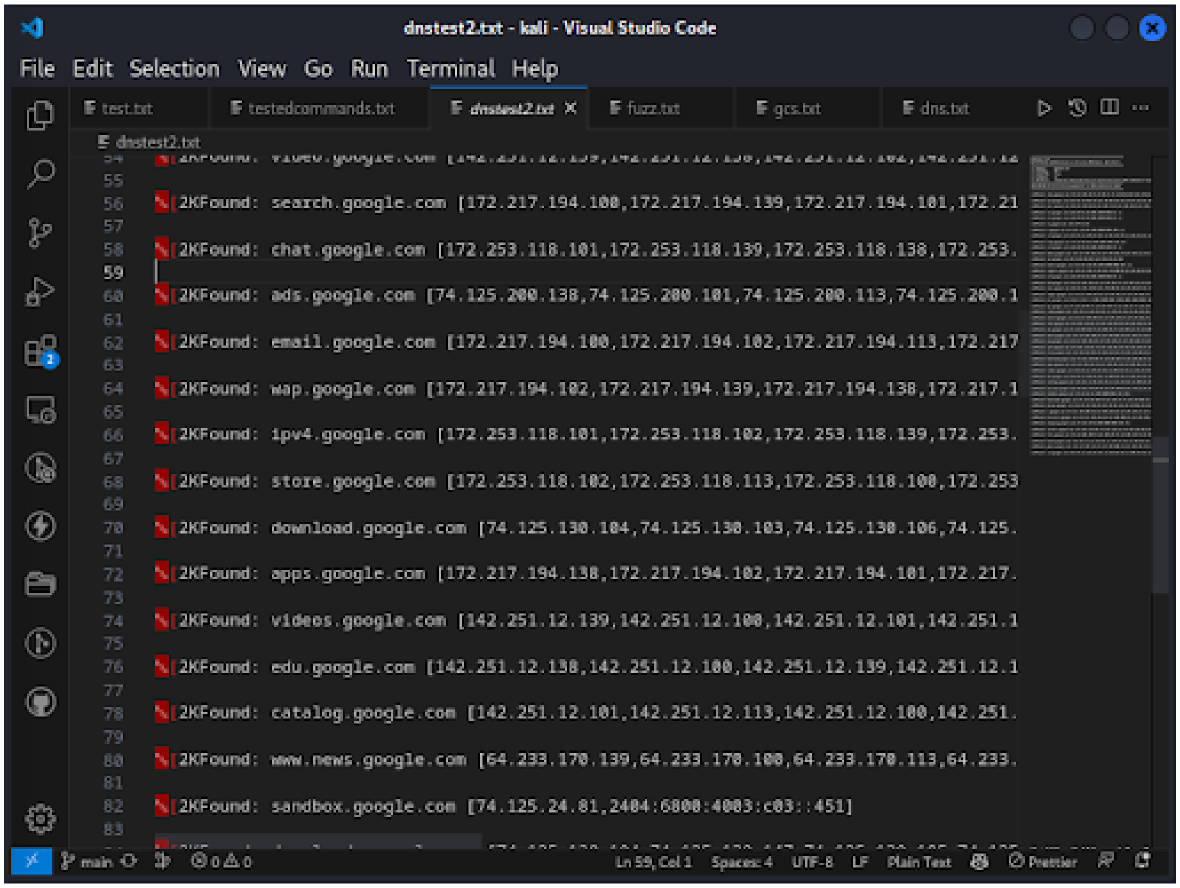
`gobuster dns -d google.com -w /subdomains-5000.txt -i --wildcard > dnstest2.txt`

Explanation of command:

- -d: used to define the domain name.

- -w: wordlist of popular subdomains.
- -i: print ips of the DNS responses.
- –wildcard: this flag is used to handle wildcard responses of dns query

The saved output is as follows:


dnstest2.txt - kali - Visual Studio Code
File Edit Selection View Go Run Terminal Help
test.txt
testedcommands.txt
dnstest2.txt
fuzz.txt
gcs.txt
dns.txt
dnstest2.txt
main
Ln 59, Col 1 Spaces: 4 UTF-8 LF Plain Text
Prettier

```

[2KFound: search.google.com [172.217.194.100,172.217.194.139,172.217.194.101,172.21

[2KFound: chat.google.com [172.253.118.101,172.253.118.139,172.253.118.138,172.253.

[2KFound: ads.google.com [74.125.200.138,74.125.200.101,74.125.200.113,74.125.200.1

[2KFound: email.google.com [172.217.194.100,172.217.194.102,172.217.194.113,172.217

[2KFound: wap.google.com [172.217.194.102,172.217.194.139,172.217.194.138,172.217.1

[2KFound: ipv4.google.com [172.253.118.101,172.253.118.102,172.253.118.139,172.253.

[2KFound: store.google.com [172.253.118.102,172.253.118.113,172.253.118.100,172.253

[2KFound: download.google.com [74.125.130.104,74.125.130.103,74.125.130.106,74.125.

[2KFound: apps.google.com [172.217.194.138,172.217.194.102,172.217.194.101,172.217.

[2KFound: videos.google.com [142.251.12.139,142.251.12.100,142.251.12.101,142.251.1

[2KFound: edu.google.com [142.251.12.138,142.251.12.100,142.251.12.139,142.251.12.1

[2KFound: catalog.google.com [142.251.12.101,142.251.12.113,142.251.12.100,142.251.

[2KFound: www.news.google.com [64.233.170.139,64.233.170.100,64.233.170.113,64.233.

[2KFound: sandbox.google.com [74.125.24.81,2404:6800:4003:c03::451]

```

Figure 24: output of the DNS command

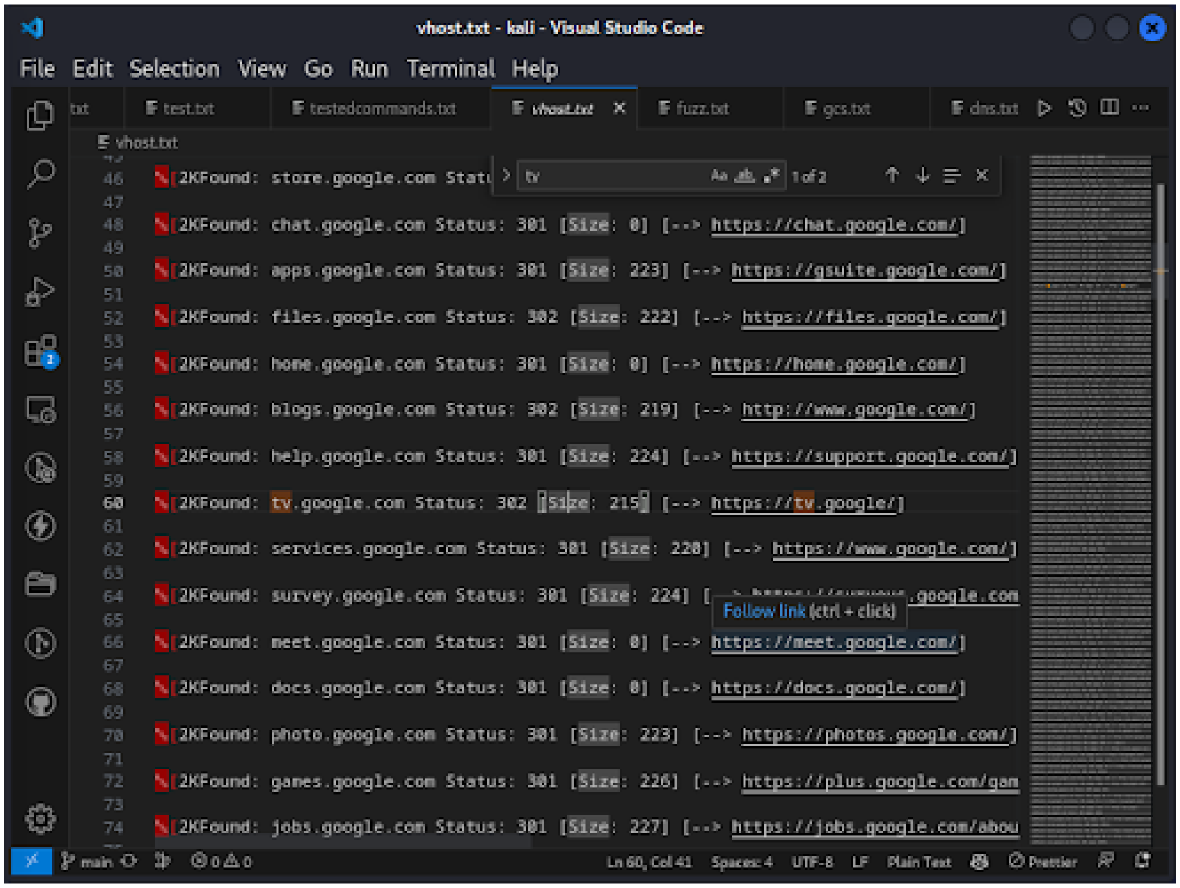
5.3 VHOST

VHOST mode will check for http responses from subdomains. In this mode we add hostnames to header and keep the original url as it is. This way the same IP can be used to host multiple websites. The basic command of VHOST is:
`gobuster vhost -u google.com -w subdomains5000.txt --append-domain > vhost.txt`
Explanation of command:

- -u: used to set the base URL.

- -w: wordlist of potential sites hosted by Google.
- –append-domain : used to add google.com to the hostname[tv.google.com]

The screenshot of the output file is given below:



```
[2KFound: store.google.com Statu

[2KFound: chat.google.com Status: 301 [Size: 0] [--> https://chat.google.com/]

[2KFound: apps.google.com Status: 301 [Size: 223] [--> https://gsuite.google.com/]

[2KFound: files.google.com Status: 302 [Size: 222] [--> https://files.google.com/]

[2KFound: home.google.com Status: 301 [Size: 0] [--> https://home.google.com/]

[2KFound: blogs.google.com Status: 302 [Size: 219] [--> http://www.google.com/]

[2KFound: help.google.com Status: 301 [Size: 224] [--> https://support.google.com/]

[2KFound: tv.google.com Status: 302 [Size: 215] [--> https://tv.google/]

[2KFound: services.google.com Status: 301 [Size: 220] [--> https://www.google.com/]

[2KFound: survey.google.com Status: 301 [Size: 224] [--> [illegible].google.com

[2KFound: meet.google.com Status: 301 [Size: 0] [--> https://meet.google.com/]

[2KFound: docs.google.com Status: 301 [Size: 0] [--> https://docs.google.com/]

[2KFound: photo.google.com Status: 301 [Size: 223] [--> https://photos.google.com/]

[2KFound: games.google.com Status: 301 [Size: 226] [--> https://plus.google.com/gam

[2KFound: jobs.google.com Status: 301 [Size: 227] [--> https://jobs.google.com/abou
```

Figure 25: output of the vhost command

We can see Google hosts multiple sites virtually, support.google.com, tv.google, gsuite etc. We can try to breach the actual host's security by first breaching the security of one of the virtual hosts that have weak security. This way attackers can use vhost mode.

5.4 FUZZ

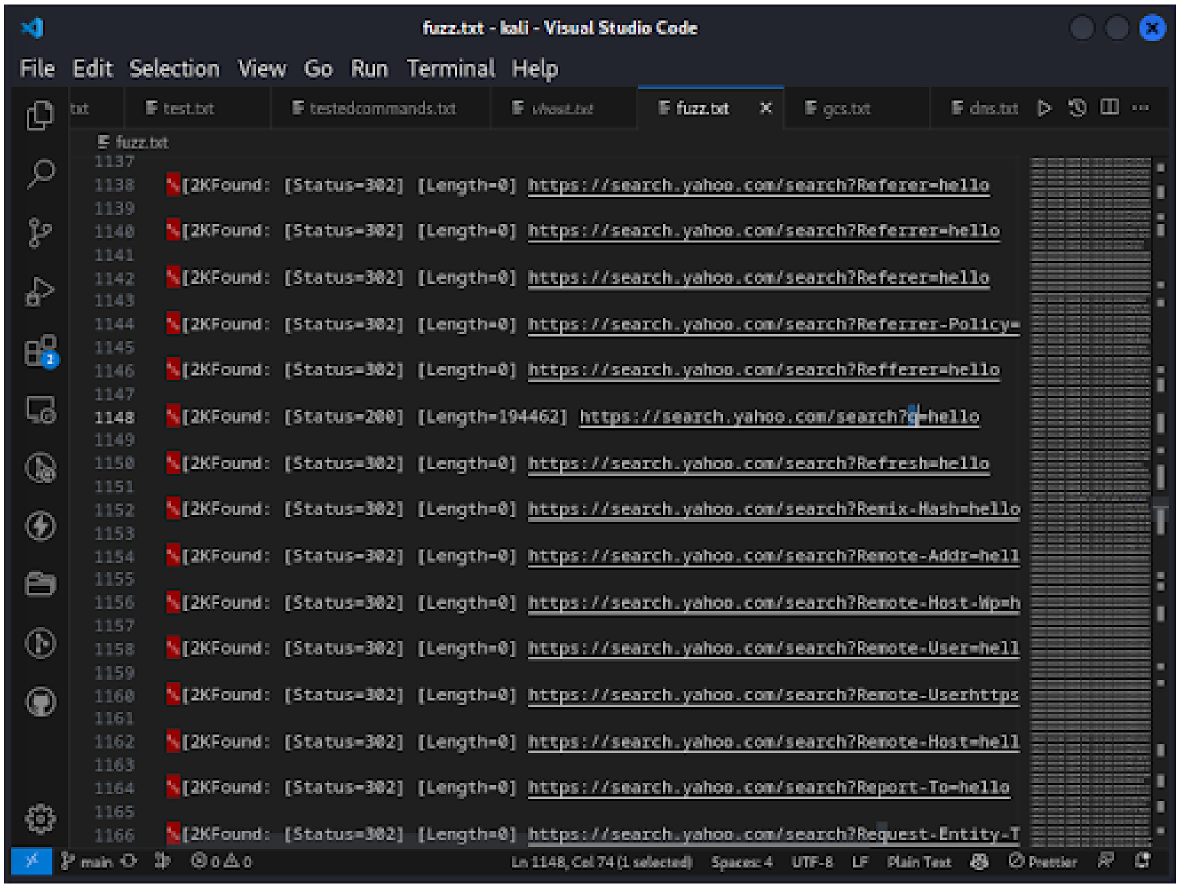
FUZZ mode can detect valid parameters of a website by brute forcing different words from wordlist and putting them in different places of the website. The basic command of FUZZ is :

```
gobuster fuzz -u search.yahoo.com/search?FUZZ=hello -w /headers -b 400,301
> fuzz.txt
```

Explanation of command:

- -u: used to set the URL we use for fuzzing. notice the URL string, we use FUZZ keywords at different places of the URL. This keyword gets replaced by words from wordlists and then the requests are sent.
- -w: wordlist of words that replace FUZZ.
- -b: blacklist status codes. we blacklist 400 and 301 to stop bad requests and redirections from occurring.

The screenshot of the output file is given below:


fuzz.txt - kali - Visual Studio Code
File Edit Selection View Go Run Terminal Help
txt
test.txt
testedcommands.txt
vhost.txt
fuzz.txt
gcs.txt
dns.txt
fuzz.txt
main
0 0
Ln 1148, Col 74 (1 selected)
Spaces: 4
UTF-8
LF
Plain Text
Prettier

```

[2KFound: [Status=302] [Length=0] https://search.yahoo.com/search?Referer=hello

[2KFound: [Status=302] [Length=0] https://search.yahoo.com/search?Referrer=hello

[2KFound: [Status=302] [Length=0] https://search.yahoo.com/search?Referer=hello

[2KFound: [Status=302] [Length=0] https://search.yahoo.com/search?Referrer-Policy=

[2KFound: [Status=302] [Length=0] https://search.yahoo.com/search?Refferer=hello

[2KFound: [Status=200] [Length=194462] https://search.yahoo.com/search?q=hello

[2KFound: [Status=302] [Length=0] https://search.yahoo.com/search?Refresh=hello

[2KFound: [Status=302] [Length=0] https://search.yahoo.com/search?Remix-Hash=hello

[2KFound: [Status=302] [Length=0] https://search.yahoo.com/search?Remote-Addr=hell

[2KFound: [Status=302] [Length=0] https://search.yahoo.com/search?Remote-Host-Wp=h

[2KFound: [Status=302] [Length=0] https://search.yahoo.com/search?Remote-User=hell

[2KFound: [Status=302] [Length=0] https://search.yahoo.com/search?Remote-Userhttps

[2KFound: [Status=302] [Length=0] https://search.yahoo.com/search?Remote-Host=hell

[2KFound: [Status=302] [Length=0] https://search.yahoo.com/search?Report-To=hello

[2KFound: [Status=302] [Length=0] https://search.yahoo.com/search?Request-Entity-T
```

Figure 26: output of the fuzz command

We can see that the keyword fuzz got replaced by q and a response generated from that request returned 200. That means q must be a valid parameter of the website. This way attackers can find parameters of a website and then inject values into parameters to launch attacks.

5.5 GCS

Google Cloud Server(GCS) Bucket is a file storage service, accessible via API call. Gobuster can enumerate over a wordlist and check if that bucket exists. The command is:

`gobuster gcs -w ./s3words.txt -t 100 > gcspartho.txt`

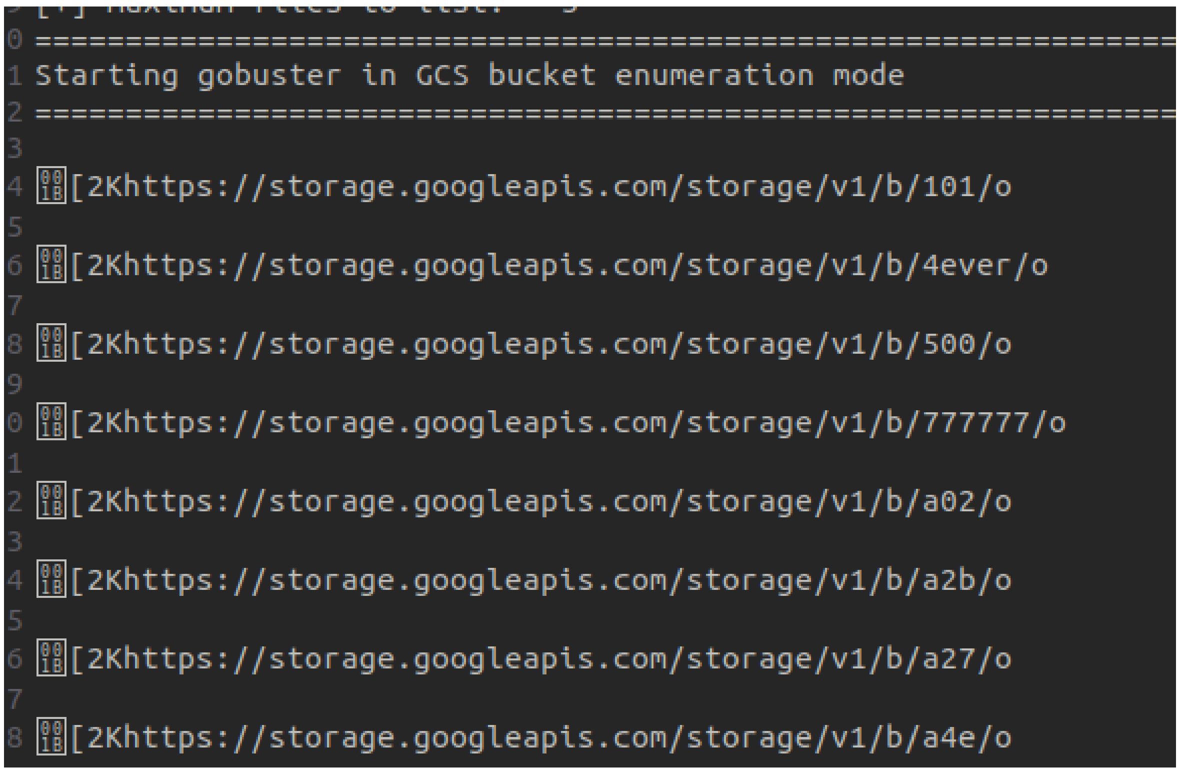
Explanation of command:

- -w: wordlist of possible Bucket name.
- -t: number of threads to be run.

Gobuster knows the API URL of GCS bucket:

`https://storage.googleapis.com/storage/v1/b/bucket/o`

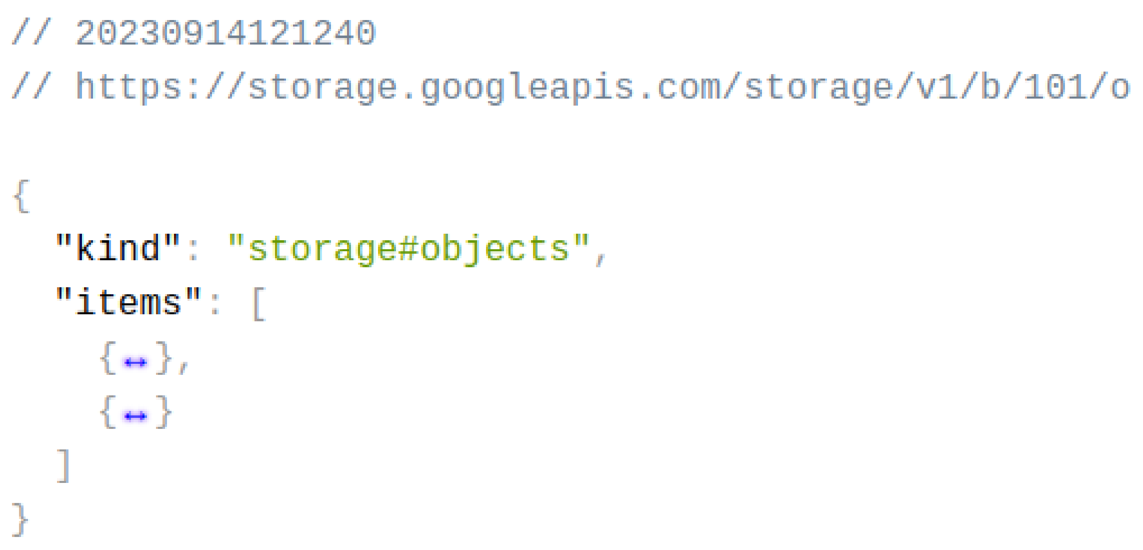
It tries to replace the bucket part with a word from the wordlist and check if it is accessible. The output looks something like this:



```
==============================================================
Starting gobuster in GCS bucket enumeration mode
==============================================================

␛[2Khttps://storage.googleapis.com/storage/v1/b/101/o

␛[2Khttps://storage.googleapis.com/storage/v1/b/4ever/o

␛[2Khttps://storage.googleapis.com/storage/v1/b/500/o

␛[2Khttps://storage.googleapis.com/storage/v1/b/777777/o

␛[2Khttps://storage.googleapis.com/storage/v1/b/a02/o

␛[2Khttps://storage.googleapis.com/storage/v1/b/a2b/o

␛[2Khttps://storage.googleapis.com/storage/v1/b/a27/o

␛[2Khttps://storage.googleapis.com/storage/v1/b/a4e/o
```

Figure 27: Portion of output after GCS Enumeration

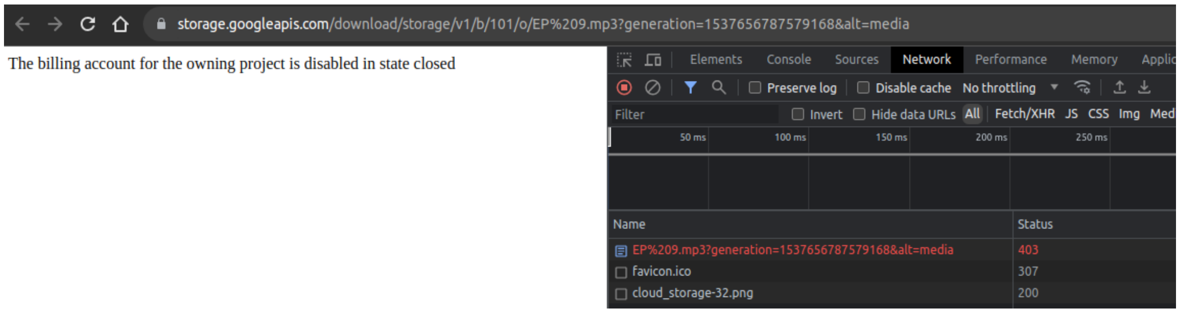
If we visit one of the links we will find a JSON object.



```
// 20230914121240
// https://storage.googleapis.com/storage/v1/b/101/o

{
  "kind": "storage#objects",
  "items": [
    {↔},
    {↔}
  ]
}
```

Figure 28: GCS API shape

The items section contains a media Link to access each one. When visited, most of them return a 403 status.



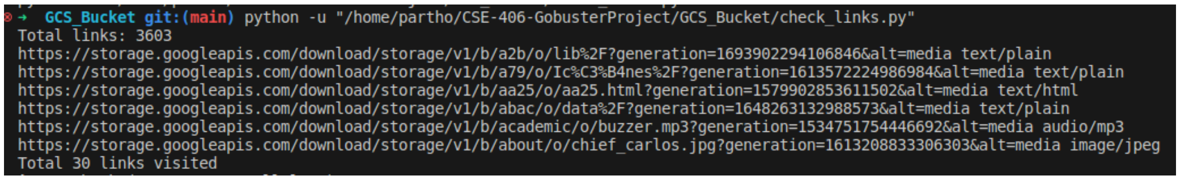
storage.googleapis.com/download/storage/v1/b/101/o/EP%209.mp3?generation=1537656787579168&alt=media

The billing account for the owning project is disabled in state closed

```
Name                                                Status
EP%209.mp3?generation=1537656787579168&alt=media    403
favicon.ico                                         307
cloud_storage-32.png                                200
```

Figure 29: Error when accessing files

So, we have created two python files (github link) to format the output file create_links.py and then iterate over each item to check if its items are accessible (check_links.py). The program will output the first item of each bucket if it is accessible. We have added some files in the Files_Found folder.

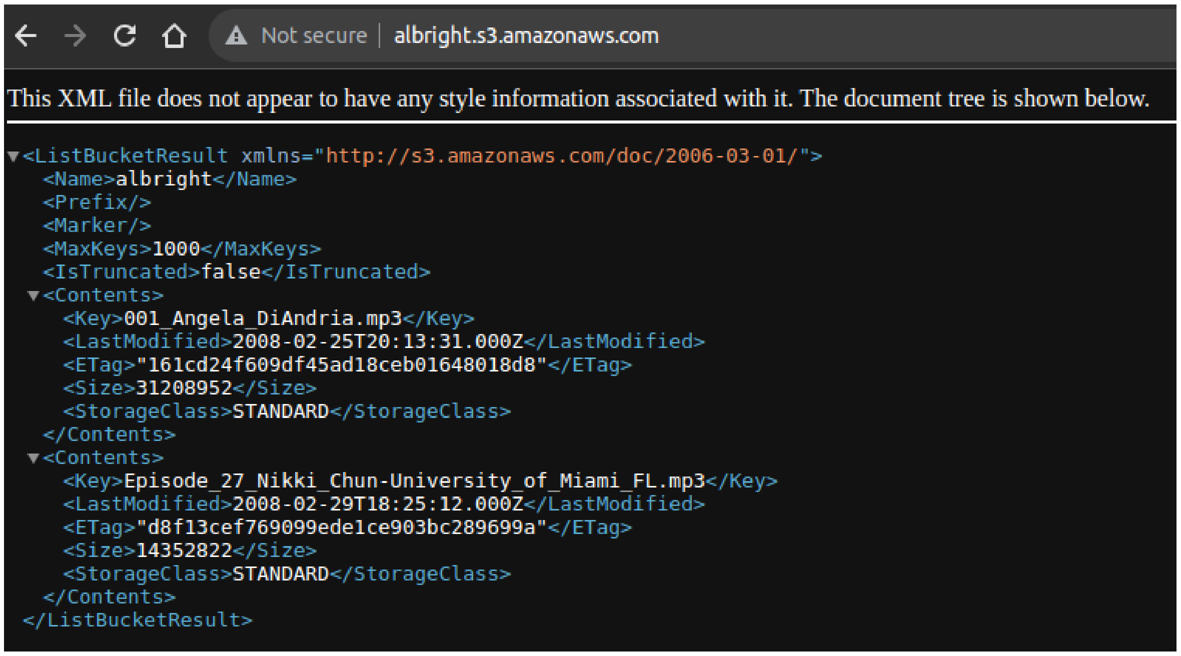


```
→  GCS_Bucket git:(main) python -u "/home/partho/CSE-406-GobusterProject/GCS_Bucket/check_links.py"
Total links: 3603
https://storage.googleapis.com/download/storage/v1/b/a2b/o/lib%2F?generation=1693902294106846&alt=media text/plain
https://storage.googleapis.com/download/storage/v1/b/a79/o/Ic%C3%B4nes%2F?generation=1613572224986984&alt=media text/plain
https://storage.googleapis.com/download/storage/v1/b/aa25/o/aa25.html?generation=1579902853611502&alt=media text/html
https://storage.googleapis.com/download/storage/v1/b/abac/o/data%2F?generation=1648263132988573&alt=media text/plain
https://storage.googleapis.com/download/storage/v1/b/academic/o/buzzer.mp3?generation=1534751754446692&alt=media audio/mp3
https://storage.googleapis.com/download/storage/v1/b/about/o/chief_carlos.jpg?generation=1613208833306303&alt=media image/jpeg
Total 30 links visited
```

Figure 30: Accessible Files found

5.6 S3

Amazon S3 bucket is similar to the GCS bucket. We need to change the mode to ”s3”.
`gobuster s3 -w ./s3words.txt -t 100 > s3output2.txt`
We get an output almost similar to a GCS bucket. Most of the links are not accessible like before. Those that are working return an XML response.



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<ListBucketResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
   <Name>albright</Name>
   <Prefix/>
   <Marker/>
   <MaxKeys>1000</MaxKeys>
   <IsTruncated>false</IsTruncated>
  ▼<Contents>
     <Key>001_Angela_DiAndria.mp3</Key>
     <LastModified>2008-02-25T20:13:31.000Z</LastModified>
     <ETag>"161cd24f609df45ad18ceb01648018d8"</ETag>
     <Size>31208952</Size>
     <StorageClass>STANDARD</StorageClass>
   </Contents>
  ▼<Contents>
     <Key>Episode_27_Nikki_Chun-University_of_Miami_FL.mp3</Key>
     <LastModified>2008-02-29T18:25:12.000Z</LastModified>
     <ETag>"d8f13cef769099ede1ce903bc289699a"</ETag>
     <Size>14352822</Size>
     <StorageClass>STANDARD</StorageClass>
   </Contents>
 </ListBucketResult>
```

Figure 31: XML Response

The key for each Contents tag can be appended after the URL to access the file. If we check the first file the URL to visit will be:
https://albright.s3.amazonaws.com/001_Angela_DiAndria.mp3.
We get a .mp3 file.

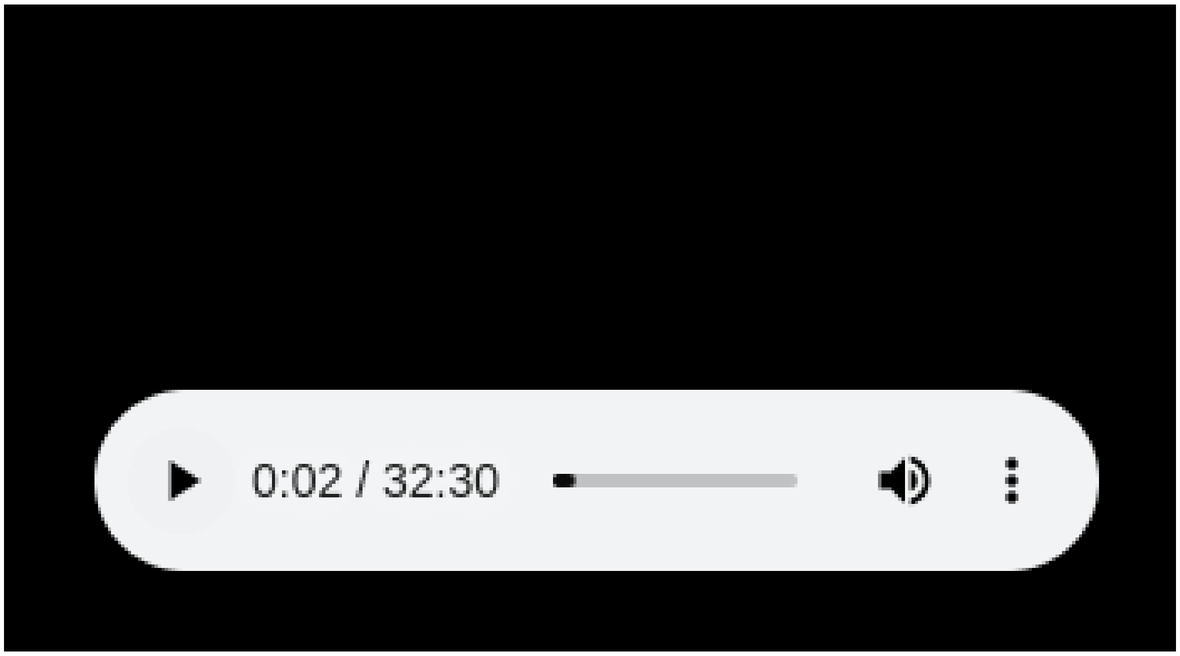
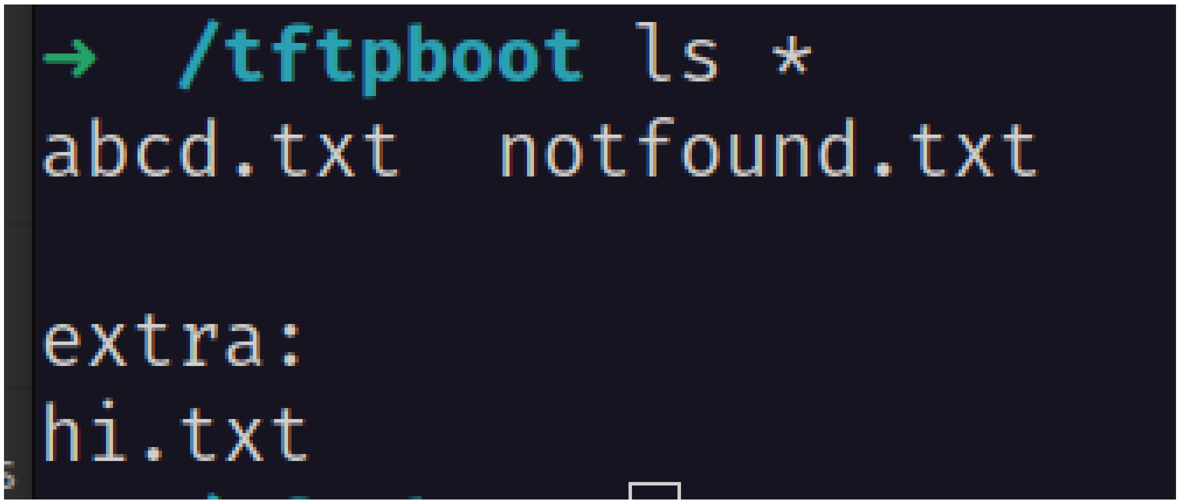


Figure 32: S3 mp3 file

5.7 tftp

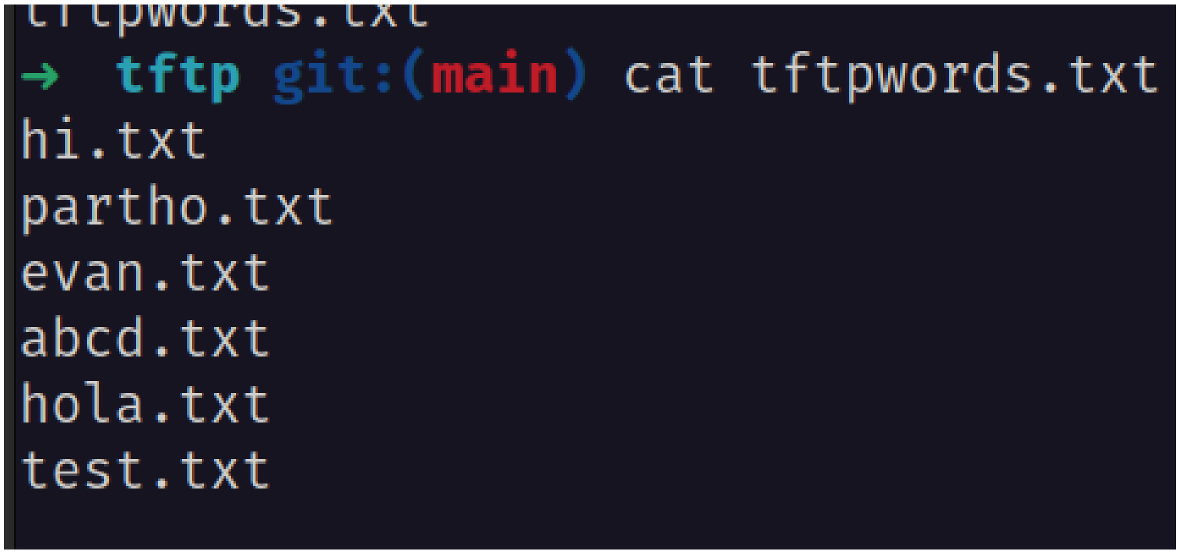
TFTP(Trivial File Transfer Protocol) is a simple file transfer protocol. Unlike FTP, it does not have any sophisticated authentication method. So, it is mostly used in a local environment. Gobuster can iterate over a wordlist and find any file existing in the root directory of the TFTP server. For testing, we first setup a TFTP server on the localhost(127.0.0.1) following this tutorial.[3] The server was setup on the **/tftpboot** directory. It had the following file structure:



```
→  /tftpboot ls *
abcd.txt  notfound.txt

extra:
hi.txt
```

Figure 33: /tftpboot File Structure

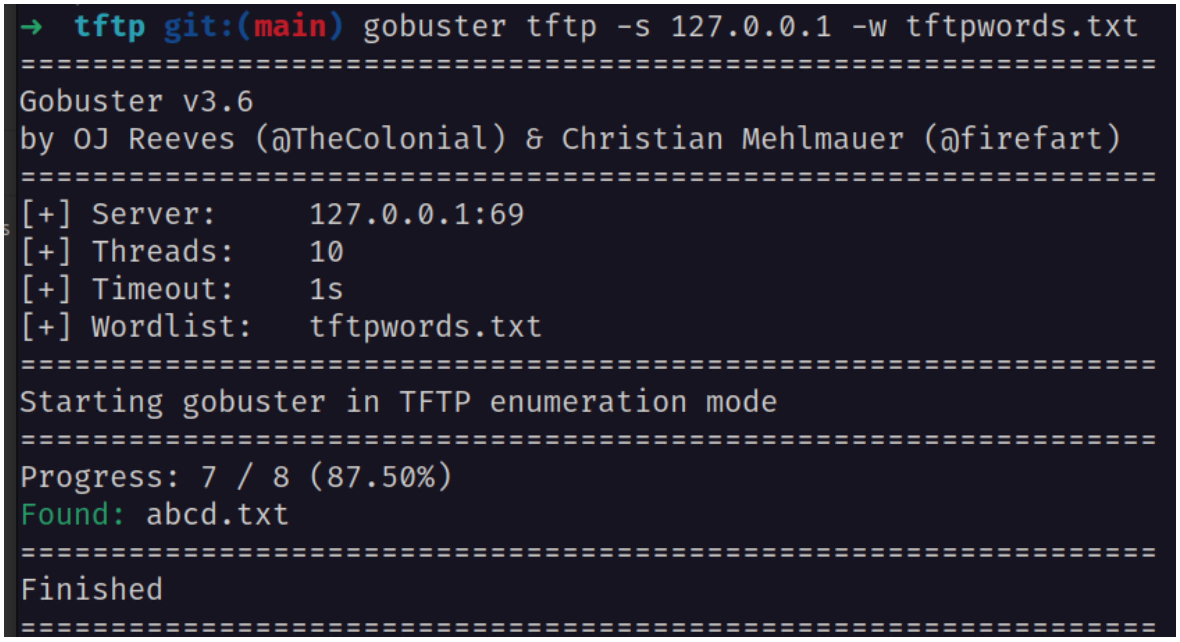
Then we can run this command: `gobuster tftp -s 127.0.0.1 -w tftpwords.txt`



```
→  tftp git:(main) cat tftpwords.txt
hi.txt
partho.txt
evan.txt
abcd.txt
hola.txt
test.txt
```

Figure 34: tftpwords.txt

We will get the following output:



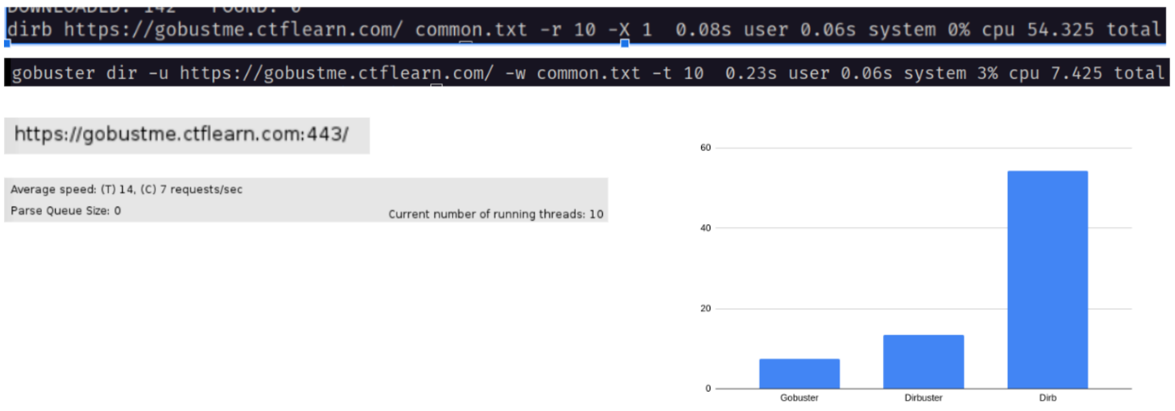
```
→  tftp git:(main) gobuster tftp -s 127.0.0.1 -w tftpwords.txt
===============================================================
Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Server:      127.0.0.1:69
[+] Threads:     10
[+] Timeout:     1s
[+] Wordlist:    tftpwords.txt
===============================================================
Starting gobuster in TFTP enumeration mode
===============================================================
Progress: 7 / 8 (87.50%)
Found: abcd.txt
===============================================================
Finished
===============================================================
```

Figure 35: tftp Output

As we can see, Gobuster found the abcd.txt, but not the other files (notFound.txt not in the wordlist, hi.txt not in the root directory.

6 Comparative Analysis

We also ran some tests to check how well Gobuster performs compared to other enumeration tools. We compared runtime for directory searching of Dirb and Dirbuster.



```
DOWNLOADED: 142    FOUND: 0
dirb https://gobustme.ctflearn.com/ common.txt -r 10 -X 1  0.08s user 0.06s system 0% cpu 54.325 total
```

```
gobuster dir -u https://gobustme.ctflearn.com/ -w common.txt -t 10  0.23s user 0.06s system 3% cpu 7.425 total
```

https://gobustme.ctflearn.com:443/

Average speed: (T) 14, (C) 7 requests/sec
Parse Queue Size: 0
Current number of running threads: 10

Figure 36: Comparative Analysis

Dirb does not support any form of threading. So, it is quite slow. Dirbuster, wriiten in Java, performs much better than Dirb. It also provides a GUI. So, this is a very beginner-friendly option. But Gobuster took almost half the time. Since, Gobuster is written in Go, a language known for its efficient concurrency feature, it should be the preferred option when running on any big wordlist.

References

[1] Github Repo `https://github.com/SheikhHasanulBanna/CSE-406-GobusterProject`

[2] SecLists (Repo with wordlists we used) `https://github.com/danielmiessler/SecLists`

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