

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	2
1.1	<u>Key Features of Gobuster</u>	2
1.2	Why use Gobuster?	2
2	Installation	3
3	Source Code Overview	3
3.1	cli	3
3.2	libgobuster	5
3.3	gobusterdir	6
3.4	gobusterdns	7
3.5	gobustervhost	8
3.6	gobusterfuzz	8
3.7	gobustergcs	9
3.8	gobusters3	10
3.9	gobustertftp	10
4	Documentation	11
4.1	DIR	12
4.2	DNS	14
4.3	VHOST	15
4.4	FUZZ	17
4.5	GCS	18
4.6	S3	20
4.7	TFTP	21
5	Demonstration	22
5.1	DIR	22
5.2	DNS	25
5.3	VHOST	26
5.4	FUZZ	26
5.5	GCS	27
5.6	S3	30
5.7	TFTP	31

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.

```
func resultWorker(g *libgobuster.Gobuster, filename string, wg *sync.WaitGroup) {
    defer wg.Done()

    var f *os.File
    var err error
    if filename != "" {
        f, err = os.Create(filename)
        if err != nil {
            g.logger.Fatalf("error on creating output file: %v", err)
        }
        defer f.Close()
    }

    for r := range g.Progress.ResultChan {
        s, err := r.ResultToString()
        if err != nil {
            g.logger.Fatalf(err)
        }
        if s != "" {
            s = strings.TrimSpace(s)
            _, err = fmt.Printf("%s\n", TERMINAL_CLEAR_LINE, s)
            if f != nil {
                err = writeToFile(f, s)
                if err != nil {
                    g.logger.Fatalf("error on writing output file: %v", err)
                }
            }
        }
    }
}
```

Figure 1: resultWorker Function

```
func errorWorker(g *libgobuster.Gobuster, wg *sync.WaitGroup) {
    defer wg.Done()

    for e := range g.Progress.ErrorChan {
        if lg.Opts.Quiet && !lg.Opts.Verbose {
            g.logger.Error(e.Error())
            g.logger.Debug("%v", e)
        }
    }

    // messageWorker outputs messages as they come in. This needs to be a range and should not handle
    // the context so the channel always has a receiver and libgobuster will not block.
    func messageWorker(g *libgobuster.Gobuster, wg *sync.WaitGroup) {
        defer wg.Done()

        for msg := range g.Progress.MessageChan {
            if lg.Opts.Quiet {
                switch msg.Level {
                    case libgobuster.LevelDebug:
                        g.logger.Debug(msg.Message)
                    case libgobuster.LevelError:
                        g.logger.Error(msg.Message)
                    case libgobuster.LevelInfo:
                        g.logger.Info(msg.Message)
                    default:
                        panic(fmt.Sprintf("invalid level %d", msg.Level))
                }
            }
        }
    }
}
```

Figure 2: errorWorker and messageWorker Function

```
func progressWorker(ctx context.Context, g *libgobuster.Gobuster, wg *sync.WaitGroup) {
    defer wg.Done()

    tick := time.NewTicker(cliProgressUpdate)

    for {
        select {
            case <-tick.C:
                printProgress(g)
            case <-ctx.Done():
                // print the final progress so we end at 100%
                printProgress(g)
                fmt.Println()
                return
        }
    }
}
```

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.

```

// gobuster is the main entry point for the CLI
func gobuster(ctx context.Context, opts *libgobuster.Options, plugin libgobuster.GobusterPlugin, log libgobuster.Logger) error {
    // sanity checks
    if opts == nil {
        return fmt.Errorf("please provide valid options")
    }

    if plugin == nil {
        return fmt.Errorf("please provide a valid plugin")
    }

    ctxCancel, cancel := context.WithCancel(ctx)
    defer cancel()

    gobuster, err := libgobuster.NewGobuster(opts, plugin, log)
    if err != nil {
        return err
    }

    if opts.Quiet {
        log.Println("quiet")
        log.Println("By @Hacker (github.com) & @Hacker (github.com)")
        log.Println("c, err := gobuster.GetConfigString()")
        if err != nil {
            return fmt.Errorf("error on creating config string: %w", err)
        }
        log.Println(c)
        log.Println("gobuster: starting gobuster in %s mode", plugin.Name())
        if opts.WordlistOffset > 0 {
            gobuster.Logger.Printf("Loading the first %d elements...", opts.WordlistOffset)
        }
    }
}

```

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.

```

// 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 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.

```

func (g *Gobuster) Run(ctx context.Context) error {
    defer close(g.Progress.Resultchan)
    defer close(g.Progress.Errorchan)
    defer close(g.Progress.Messagechan)

    if err := g.plugin.PreRun(ctx, g.Progress); err != nil {
        return err
    }

    var workerGroup sync.WaitGroup
    workerGroup.Add(g.Opts.Threads)

    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)
    }

    scanner, err := g.getWordlist()
    if err != nil {
        return err
    }
}

```

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.

```

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 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.

```

func NewHTTPClient(opt *Options) (*HTTPClient, error) {
    var proxyURLFunc func(*http.Request) (*url.URL, error)
    var client *http.Client
    proxyURLFunc = http.ProxyFromEnvironment

    if opt == nil {
        return nil, fmt.Errorf("options is nil")
    }

    if opt.Proxy != "" {
        proxyURL, err := url.Parse(opt.Proxy)
        if err != nil {
            return nil, fmt.Errorf("proxy url is invalid (%w)", err)
        }
        proxyURLFunc = http.ProxyURL(proxyURL)
    }

    var redirectFunc func(req *http.Request, via []*http.Request) error
    if opt.FollowRedirect {
        redirectFunc = func(req *http.Request, via []*http.Request) error {
            return http.ErrUseLastResponse
        }
    } else {
        redirectFunc = nil
    }

    tlsConfig := tls.Config{
        InsecureSkipVerify: opt.NoTLSValidation,
        // enable TLS1.0 and TLS1.3 support
        MinVersion: tls.VersionTLS10,
    }
}

```

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.

```

func (g *GobusterDir) ProcessWord(ctx context.Context, word string, progress *linguist.Progress) error {
    suffix := ""
    if d.options.useSlash {
        suffix = "/"
    }
    entity := fmt.Sprintf("%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 rid of first char and trim it
        entity = strings.TrimLeft(entity, "/")
    }
    url := fmt.Sprintf("%s%s", d.options.url, entity)

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

```

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.

```

327 var http int64
328 var header http.Header
329 for i := 1; i <= tries; i++ {
330     var err error
331     statusCode, http.Header, _, err = d.http.Request(ctx, url, libpuster.RequestOptions())
332     if err != nil {
333         // Check if it's a timeout and if we should try again and try again
334         // otherwise the timeout error is raised
335         if netErr, ok := err.(net.Error); ok && netErr.Timeout() && i <= tries {
336             continue
337         } else if strings.Contains(err.Error(), "invalid control character in url") {
338             // the error is raised when we try to print out and ignore it
339             // so gobuster will not exit
340             progress.FinishChan <- err
341             continue
342         } else {
343             return err
344         }
345     }
346     break
347 }
348 if statusCode != 0 {
349     resultStatus := false
350     if d.options.StatusCodeBlacklistParsed.Length() > 0 {
351         if d.options.StatusCodeBlacklistParsed.Contains(statusCode) {
352             resultStatus = true
353         }
354     } else if d.options.StatusCodeWhitelistParsed.Length() > 0 {
355         if d.options.StatusCodeWhitelistParsed.Contains(statusCode) {
356             resultStatus = true
357         }
358     } else {
359         return fmt.Errorf("StatusCode and StatusCodeList are both not set which should not happen")
360     }
361 }
362 }

```

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:

```

func (d *GobusterDNS) ProcessWord(ctx context.Context, word string, progress *libpuster.Progress) error {
    subdomain := fmt.Sprintf("%s.%s", word, d.options.Domain)
    if d.options.Verbose && strings.HasPrefix(subdomain, ".") {
        // and 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.dns.Lookup(ctx, subdomain)
    if err != nil {
        if !d.isWildcard() || !d.wildcardIps.ContainsAny(ips) {
            result := Result{
                Found:      false,
                ShowIPs:    d.options.ShowIPs,
                ShowName:    d.options.ShowName,
                Name:        d.options.Name,
            }
            if d.options.ShowName {
                result.IPs = ips
            } else if d.options.ShowName {
                name, err := d.dns.LookupName(ctx, subdomain)
                if err != nil {
                    result.Name = name
                }
            }
            progress.ResultChan <- result
        } else if d.options.Verbose {
            progress.ResultChan <- Result{
                Found:      false,
                ShowIPs:    d.options.ShowIPs,
                ShowName:    d.options.ShowName,
            }
        }
    }
    return nil
}

```

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 **dns.Lookup** 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.

```
func (v *GobusterVhost) ProcessWord(ctx context.Context, word string, progress *libbustervhost.Progress) error {
    var subdomain string
    if v.options.AppendDomain {
        subdomain = fmt.Sprintf("%s.%s", word, v.domain)
    } else {
        // wordlist needs to include full domain
        subdomain = word
    }

    tries := 1
    if v.options.RetryTimeout && v.options.RetryAttempts > 0 {
        // add 1 to 1 will be the overall max requests
        tries += v.options.RetryAttempts
    }

    for i := 1; i <= tries; i++ {
        // send http request
        statusCode, size, header, body, err := v.httpRequest(ctx, v.options.URL, libbustervhost.RequestOptions{Host: subdomain, RetryBody:
            if err != nil {
                // check if it's a timeout and if we should try again and try again
                // otherwise the timeout error is returned
                if netErr, ok := err.(net.Error); ok && netErr.Timeout() && i != tries {
                    continue
                } else if strings.Contains(err.Error(), "invalid control character in url") {
                    // not error to error than so it's printed out and ignore it
                    // (it generates null and null)
                    progress.ResultChan <- err
                    continue
                } else {
                    return err
                }
            }
        }
    }
    break
}
```

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.

```
var statusCode int
var size int64
var header http.Header
var body []byte
for i := 1; i <= tries; i++ {
    // send http request
    statusCode, size, header, body, err := v.httpRequest(ctx, v.options.URL, libbustervhost.RequestOptions{Host: subdomain, RetryBody:
        if err != nil {
            // check if it's a timeout and if we should try again and try again
            // otherwise the timeout error is returned
            if netErr, ok := err.(net.Error); ok && netErr.Timeout() && i != tries {
                continue
            } else if strings.Contains(err.Error(), "invalid control character in url") {
                // not error to error than so it's printed out and ignore it
                // (it generates null and null)
                progress.ResultChan <- err
                continue
            } else {
                return err
            }
        }
    }
    break
}
```

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.

```
// subdomain must not match default vhost and non-existent vhost
// or verbose mode is enabled
found := body != nil && !bytes.Equal(body, v.normalBody) && !bytes.Equal(body, v.abnormalBody)
if (found && !v.options.IncludeLengthHeader.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 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.

```
func (d *gobusterFuzz) ProcessWord(ctx context.Context, word string, progress *gobusterProgress) 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),
            }
        }
    }

    if d.options.RequestBody != "" {
        data := strings.ReplaceAll(d.options.RequestBody, FuzzKeyword, word)
        buffer := strings.NewReader(data)
        requestOptions.Body = buffer
    }

    // fuzzing of basic auth
    if strings.Contains(d.options.Username, FuzzKeyword) || strings.Contains(d.options.Password, FuzzKeyword) {
        requestOptions.UpdateBasicAuthUsername = strings.ReplaceAll(d.options.Username, FuzzKeyword, word)
        requestOptions.UpdateBasicAuthPassword = strings.ReplaceAll(d.options.Password, FuzzKeyword, word)
    }

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

Figure 15: Constructing URL in fuzz mode

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

```
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
}

if statusCode != 0 {
    resultStatus = true
    if d.options.ExcludeLengthParsed.Contains(int(size)) {
        resultStatus = false
    }

    if d.options.ExcludeStatusCodesParsed.Length() > 0 {
        if d.options.ExcludeStatusCodesParsed.Contains(statusCode) {
            resultStatus = false
        }
    }
}
```

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.

```
func (s *gobusters3) processWord(ctx context.Context, word string, progress *libgobuster.Progress) error {
    // only check for valid bucket names
    if !IsValidBucketName(word) {
        return nil
    }

    bucketURL := fmt.Sprintf("https://storage.googleapis.com/storage/v1/b/%s/o?recursive=true", word, s.options.HostInfo.Host)

    tries := 1
    if s.options.RetryTimeout < s.options.RetryAttempts > 0 {
        // add it so it will be the correct number of requests
        tries += s.options.RetryAttempts
    }

    var statusCode int
    var body []byte
    for i := 1; i <= tries; i++ {
        var err error
        statusCode, _, body, err = s.HttpRequest(ctx, bucketURL, libgobuster.RequestOptions{ResponseBody: 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 chain so it's printed out and ignore it
                // no gobuster will not quit
                progress.ErrorChain <- err
                continue
            } else {
                return err
            }
        }
        break
    }
}
```

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.

```
func (s *gobusters3) processWord(ctx context.Context, word string, progress *libgobuster.Progress) error {
    // only check for valid bucket names
    if !IsValidBucketName(word) {
        return nil
    }

    bucketURL := fmt.Sprintf("https://s3.amazonaws.com/%s/%s", word, s.options.HostInfo.Host)

    tries := 1
    if s.options.RetryTimeout < s.options.RetryAttempts > 0 {
        // add it so it will be the correct number of requests
        tries += s.options.RetryAttempts
    }

    var statusCode int
    var body []byte
    for i := 1; i <= tries; i++ {
        var err error
        statusCode, _, body, err = s.HttpRequest(ctx, bucketURL, libgobuster.RequestOptions{ResponseBody: 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 chain so it's printed out and ignore it
                // no gobuster will not quit
                progress.ErrorChain <- err
                continue
            } else {
                return err
            }
        }
        break
    }
}
```

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.

```
// Processword is the process implementation of gobusterTftp
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")
    if err != nil {
        // file not found
        if d.globopts.Verbose {
            progress.ResultChan <- Result{
                filename: word,
                found:      false,
                errorMessage: err.Error(),
            }
        }
        return nil
    }
    result := Result{
        filename: word,
        found:      true,
    }
    if n, ok := wt.(tftp.IncomingTransfer).Size(); ok {
        result.Size = n
    }
    progress.ResultChan <- result
    return nil
}
```

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
```

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

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
```

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

Global Flags:

<code>--debug</code>	Enable debug output
<code>--delay duration</code>	Time each thread waits between requests (e.g. 1500ms)
<code>--no-color</code>	Disable color output

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

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:

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

Global Flags:

<code>--debug</code>	Enable debug output
----------------------	---------------------

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

<code>--append-domain</code>	Append main domain from URL to words from wordlist. Otherwise the fully qualified domains need to be specified in the wordlist.
<code>--client-cert-p12 string</code>	a p12 file to use for options TLS client certificates
<code>--client-cert-p12-password string</code>	the password to the p12 file
<code>--client-cert-pem string</code>	public key in PEM format for optional TLS client certificates
<code>--client-cert-pem-key string</code>	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")
```

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

Global Flags:

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

```

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

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:

```

=====
[*] Url: https://cse.buet.ac.bd
[*] Method: GET
[*] Threads: 25
[*] Wordlist: small.txt
[*] Negative status codes: 404
[*] User Agent: gobuster/3.6
[*] Expanded: true
[*] Timeout: 10s
=====
Starting gobuster in directory enumeration mode
=====
[[20https://cse.buet.ac.bd/.htaccess (Status: 403) [Size: 200]
[[20https://cse.buet.ac.bd/.hta (Status: 403) [Size: 200]
[[20https://cse.buet.ac.bd/.gitattributes (Status: 403) [Size: 200]
[[20https://cse.buet.ac.bd/.gitignore (Status: 403) [Size: 200]
[[20https://cse.buet.ac.bd/.htpasswd (Status: 403) [Size: 200]
[[20https://cse.buet.ac.bd/admin (Status: 307) [Size: 0] [--> https://cse.buet.ac.bd/login]
[[20https://cse.buet.ac.bd/ (Status: 400) [Size: 1134]
[[20https://cse.buet.ac.bd/contribute.json (Status: 403) [Size: 200]
[[20https://cse.buet.ac.bd/ht (Status: 301) [Size: 315] [--> https://cse.buet.ac.bd/ht/]
[[20https://cse.buet.ac.bd/index.php (Status: 200) [Size: 32092]
[[20https://cse.buet.ac.bd/moodle (Status: 301) [Size: 319] [--> https://cse.buet.ac.bd/moodle/]
[[20https://cse.buet.ac.bd/new (Status: 301) [Size: 316] [--> https://cse.buet.ac.bd/new/]
[[20https://cse.buet.ac.bd/option (Status: 307) [Size: 1] [--> https://cse.buet.ac.bd/login]
[[20https://cse.buet.ac.bd/research (Status: 301) [Size: 321] [--> https://cse.buet.ac.bd/research/]
[[20https://cse.buet.ac.bd/javascript (Status: 301) [Size: 323] [--> https://cse.buet.ac.bd/javascript/]
=====
Finished
  
```

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:

```

=====
Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
=====
[+] Url: https://cse.buet.ac.bd/javascript
[+] Method: GET
[+] Threads: 25
[+] Wordlist: small.txt
[+] Negative Status codes: 404
[+] User Agent: gobuster/3.6
[+] Expanded: true
[+] Timeout: 10s
=====
Starting gobuster in directory enumeration mode
=====

[[2Khttps://cse.buet.ac.bd/javascript/.htpasswd (Status: 403) [Size: 280]
[[2Khttps://cse.buet.ac.bd/javascript/.htaccess (Status: 403) [Size: 280]
[[2Khttps://cse.buet.ac.bd/javascript/.hta (Status: 403) [Size: 280]
[[2Khttps://cse.buet.ac.bd/javascript/jquery (Status: 301) [Size: 330] [--> https://cse.buet.ac.bd/javascript/jquery/]

=====
Finished
=====

```

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:

```

=====
Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
=====
[+] Url: https://cse.buet.ac.bd/javascript/jquery
[+] Method: GET
[+] Threads: 25
[+] Wordlist: small.txt
[+] Negative Status codes: 404
[+] User Agent: gobuster/3.6
[+] Expanded: true
[+] Timeout: 10s
=====
Starting gobuster in directory enumeration mode
=====

[[2Khttps://cse.buet.ac.bd/javascript/jquery/.htaccess (Status: 403) [Size: 280]
[[2Khttps://cse.buet.ac.bd/javascript/jquery/.hta (Status: 403) [Size: 280]
[[2Khttps://cse.buet.ac.bd/javascript/jquery/.htpasswd (Status: 403) [Size: 280]
[[2Khttps://cse.buet.ac.bd/javascript/jquery/jquery (Status: 200) [Size: 271809]

=====
Finished
=====

```

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:


```

1 Gobuster v3.6
2 by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
3
4 [-] Url: https://gobustme.ctflearn.com/
5 [-] Method: GET
6 [-] Threads: 25
7 [-] Wordlist: common1.txt
8 [-] Negative Status codes: 404
9 [-] User Agent: gobuster/3.6
10 [-] Expanded: true
11 [-] Timeout: 10s
12
13 Starting gobuster in directory enumeration mode
14
15 [2K]https://gobustme.ctflearn.com/.well-known/http-opportunistic (Status: 200) [Size: 32]
16
17 [2K]https://gobustme.ctflearn.com/call (Status: 301) [Size: 169] [→ http://gobustme.ctflearn.com/call/]
18
19 [2K]https://gobustme.ctflearn.com/carpet (Status: 301) [Size: 169] [→ http://gobustme.ctflearn.com/carpet/]
20
21 [2K]https://gobustme.ctflearn.com/flag (Status: 301) [Size: 169] [→ http://gobustme.ctflearn.com/flag/]
22
23 [2K]https://gobustme.ctflearn.com/flag (Status: 301) [Size: 169] [→ http://gobustme.ctflearn.com/flag/]
24
25 [2K]https://gobustme.ctflearn.com/flag (Status: 301) [Size: 169] [→ http://gobustme.ctflearn.com/flag/]
26
27 [2K]https://gobustme.ctflearn.com/index.html (Status: 200) [Size: 2713]
28
29 [2K]https://gobustme.ctflearn.com/shadow (Status: 301) [Size: 169] [→ http://gobustme.ctflearn.com/shadow/]
30
31 [2K]https://gobustme.ctflearn.com/skin (Status: 301) [Size: 169] [→ http://gobustme.ctflearn.com/skin/]
32
33 Finished
34
35
36

```

Figure 25: output of scanning gobustme

We can access [Flag hidden Site](#) and get the flag.

The flag: CTFlearngh0sbu5t3rs₄ever

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
=====

[2K]Found: moodle.cse.buet.ac.bd

[2K]Found: 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 subdomains.

The Command:

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

The Output:

```
-C:\Users\NIGHT>BurtingBuster.exe results.txt -Mmapcat

File Edit Search View Document Help

[+] p0c0n.txt x csc.txt x vhostresults.txt x

1
2 g0bst0ut vsh.k
3 by 63 H0v3r5 (@ph0c0n) & Christian M0hlmaier (@irrefart)
4
5 [3] Url: http://google.com
6 [3] Method: GET
7 [3] Threads: 25
8 [3] Delay: randomize:gapfillin=5000.txt
9 [3] User Agent: g0bst0ut/v.s.k
10 [3] Timeout: 50s
11 [3] Append domain: true
12
13 Starting g0bst0ut in VHOST enumeration mode
14
15
16 [2f]Found: dev.google.com Status: 301 [Size: 272] [-> https://developers.google.com/]
17
18 [2f]Found: m.google.com Status: 302 [Size: 282] [-> https://get.google.com/apps/apps?source=gsmleibute_campaign=redirect]
19
20 [2f]Found: blog.google.com Status: 302 [Size: 231] [-> https://www.blog.google/]
21
22 [2f]Found: www.google.com Status: 200 [Size: 20427]
23
24 [2f]Found: mobile.google.com Status: 301 [Size: 226] [-> https://www.google.com/mobile/]
25
26 [2f]Found: news.google.com Status: 301 [Size: 0] [-> https://news.google.com/]
27
28 [2f]Found: mail.google.com Status: 301 [Size: 388] [-> /mail/]
29
30 [2f]Found: support.google.com Status: 301 [Size: 224] [-> https://support.google.com/]
31
32 [2f]Found: admin.google.com Status: 301 [Size: 360] [-> http://www.google.com/very/index?cont=linux=http://admin.google.com/tq-eqglb0zJGCMQm0ZjWkVwK9tGkgUyxtfJlRvE9KwH0zb0q0Ltbx5J7zNzkyh0gYl5kZ2wHyfJspJ22skIab0p]
33
34 [2f]Found: search.google.com Status: 301 [Size: 228] [-> https://www.google.com/]
35
36 [2f]Found: video.google.com Status: 301 [Size: 218] [-> http://www.google.com/videohp]
37
38 [2f]Found: images.google.com Status: 200 [Size: 39838]
39
40 [2f]Found: ads.google.com Status: 301 [Size: 0] [-> https://ads.google.com/]
41
42 [2f]Found: wap.google.com Status: 302 [Size: 228] [-> https://wap.google.com/]
43
```

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:

```

267round: [Status=403] [length=15683] [word=Accepted] https://play.picoctf.org/practice/accepted-1
268round: [Status=403] [length=15754] [word=Accept-Language] https://play.picoctf.org/practice/accept-language-1
269round: [Status=403] [length=15872] [word=Access-Control-Allow-Credentials] https://play.picoctf.org/practice/access-control-allow-c
270round: [Status=200] [length=2466] [word=Accept-Application] https://play.picoctf.org/practice/accept-application-1
271round: [Status=403] [length=15813] [word=Access-Control-Allow-Headers] https://play.picoctf.org/practice/access-control-allow-head
272round: [Status=200] [length=2466] [word=Access-Control-Allow-Methods] https://play.picoctf.org/practice/access-control-allow-method
273round: [Status=200] [length=2466] [word=Access-Control-Request-Headers] https://play.picoctf.org/practice/access-control-request-he
274round: [Status=200] [length=2466] [word=Accept-Encoding] https://play.picoctf.org/practice/accept-encoding-1
275round: [Status=200] [length=2466] [word=Accept-Ranges] https://play.picoctf.org/practice/accept-ranges-1
276round: [Status=200] [length=2466] [word=Accept-Encoding] https://play.picoctf.org/practice/accept-encoding-1
277round: [Status=200] [length=2466] [word=Accept-Charset] https://play.picoctf.org/practice/accept-charset-1
278round: [Status=200] [length=2466] [word=Accept-Version] https://play.picoctf.org/practice/accept-version-1
279round: [Status=200] [length=2466] [word=Accesskey] https://play.picoctf.org/practice/accesskey-1
280round: [Status=200] [length=2466] [word=Access-Control-Allow-Origin] https://play.picoctf.org/practice/access-control-allow-origin
281round: [Status=200] [length=2466] [word=Action] https://play.picoctf.org/practice/action-1
282round: [Status=200] [length=2466] [word=Admin] https://play.picoctf.org/practice/admin-1
283round: [Status=200] [length=2466] [word=Access-Control-Expose-Headers] https://play.picoctf.org/practice/access-control-expose-head
284round: [Status=200] [length=2466] [word=Age] https://play.picoctf.org/practice/age-1

```

Figure 27: Output of the fuzz mode

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

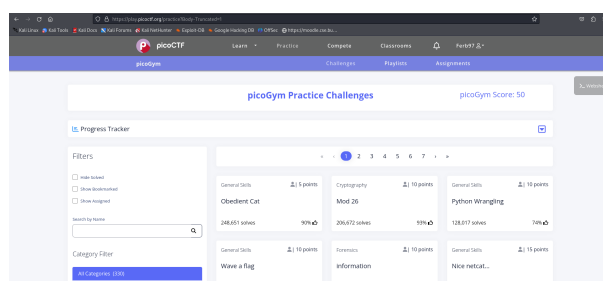


Figure 28: Output result of the fuzz mode

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:

```

10
11 Starting gobuster in GCS bucket enumeration mode
12
13
14 [2K]https://storage.googleapis.com/storage/v1/b/101/o
15
16 [2K]https://storage.googleapis.com/storage/v1/b/4ever/o
17
18 [2K]https://storage.googleapis.com/storage/v1/b/500/o
19
20 [2K]https://storage.googleapis.com/storage/v1/b/777777/o
21
22 [2K]https://storage.googleapis.com/storage/v1/b/a02/o
23
24 [2K]https://storage.googleapis.com/storage/v1/b/a12345678/o
25
26 [2K]https://storage.googleapis.com/storage/v1/b/a4e/o
27
28 [2K]https://storage.googleapis.com/storage/v1/b/a79/o
29
30 [2K]https://storage.googleapis.com/storage/v1/b/aa11/o
31
32 [2K]https://storage.googleapis.com/storage/v1/b/aa25/o
33
34 [2K]https://storage.googleapis.com/storage/v1/b/aadhaar/o
35
36 [2K]https://storage.googleapis.com/storage/v1/b/aagc/o
37
38 [2K]https://storage.googleapis.com/storage/v1/b/abbas/o
39
40 [2K]https://storage.googleapis.com/storage/v1/b/abcbac/o
41
42 [2K]https://storage.googleapis.com/storage/v1/b/abcd1234/o
43
44 [2K]https://storage.googleapis.com/storage/v1/b/ability/o

```

Figure 29: Output of the gcs mode

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

```

{
  "kind": "storage#object",
  "id": "13176547879368",
  "mediaLink": "https://www.googleapis.com/download/storage/v1/b/101/o/13176547879368?generation=15176547879368&alt=media",
  "name": "101",
  "bucket": "101",
  "generation": "15176547879368",
  "contentType": "text/html",
  "storageClass": "REGIONAL",
  "size": "4197280",
  "md5hash": "6c9c9b3a3c9a8a810d0e",
  "crc32c": "9b41f0e",
  "etag": "CKQdP72WZC5C+",
  "timeCreated": "2018-09-20T22:53:47.578Z",
  "updated": "2018-09-20T22:53:47.578Z",
  "timeStorageClassUpdated": "2018-09-20T22:53:47.578Z"
}

```

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.

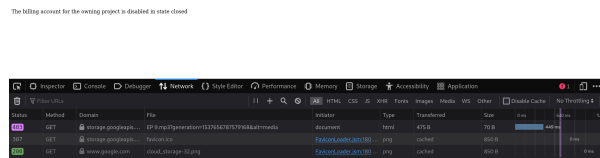


Figure 31: Failed Output of the gcs mode

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.

```

1 import requests
2 import re
3
4 # Read from output file
5 with open("gcs1.txt", "r") as txt:
6     string = txt.read()
7
8 # Format links
9 links = re.findall(r'https://\[^\\s\]+', string)
10
11 # Write to output file
12 with open("formatted_links.txt", "w") as output:
13     for link in links:
14         output.write(link+"\n")
15

```

Figure 32: Code for formatting the gcs output

```

# find working links
with open("formatted.links.txt", "w") 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)

            # Son 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("content_type")
                    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")
                        linkfile.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.

```

1 https://storage.googleapis.com/storage/v1/b/100/f/
2 https://storage.googleapis.com/storage/v1/b/10over/f/
3 https://storage.googleapis.com/storage/v1/b/1900/f/
4 https://storage.googleapis.com/storage/v1/b/1777777/f/
5 https://storage.googleapis.com/storage/v1/b/180/f/
6 https://storage.googleapis.com/storage/v1/b/13145678/f/
7 https://storage.googleapis.com/storage/v1/b/140/f/
8 https://storage.googleapis.com/storage/v1/b/149/f/
9 https://storage.googleapis.com/storage/v1/b/150/f/
10 https://storage.googleapis.com/storage/v1/b/152/f/
11 https://storage.googleapis.com/storage/v1/b/aadhaar/f/
12 https://storage.googleapis.com/storage/v1/b/aaef/f/
13 https://storage.googleapis.com/storage/v1/b/abbas/f/
14 https://storage.googleapis.com/storage/v1/b/abcdef/f/
15 https://storage.googleapis.com/storage/v1/b/abcxyz/f/
16 https://storage.googleapis.com/storage/v1/b/ability/f/
17 https://storage.googleapis.com/storage/v1/b/abusement/f/
18 https://storage.googleapis.com/storage/v1/b/about/f/
19 https://storage.googleapis.com/storage/v1/b/academi/f/
20 https://storage.googleapis.com/storage/v1/b/academic/f/
21 https://storage.googleapis.com/storage/v1/b/accessed/f/
22 https://storage.googleapis.com/storage/v1/b/accident/f/
23 https://storage.googleapis.com/storage/v1/b/accidents/f/
24 https://storage.googleapis.com/storage/v1/b/accessibility/f/
25 https://storage.googleapis.com/storage/v1/b/accent/f/
26 https://storage.googleapis.com/storage/v1/b/accept/f/
27 https://storage.googleapis.com/storage/v1/b/ad1/f/
28 https://storage.googleapis.com/storage/v1/b/adamant/f/
29 https://storage.googleapis.com/storage/v1/b/addressbook/f/
30 https://storage.googleapis.com/storage/v1/b/addressee/f/
31 https://storage.googleapis.com/storage/v1/b/adm1n-levy/f/
32 https://storage.googleapis.com/storage/v1/b/adm1ns/f/
33 https://storage.googleapis.com/storage/v1/b/adm-in-staging/o
34 https://storage.googleapis.com/storage/v1/b/adoption/f/
35 https://storage.googleapis.com/storage/v1/b/adviser/f/
36 https://storage.googleapis.com/storage/v1/b/advs/f/

```

Figure 34: Links after formatting the gcs output

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

```
--<Error>
  <Code>AccessDenied</Code>
  <Message>Access Denied</Message>
  <RequestId>Z3RGEBVGZB2D6RKR</RequestId>
--<HostId>
  G30q02GNCDBW/PFve/bPZUSciU2hMeFK8sRcqYrNuR0I/zr6f9v6ft3QOgSMhkHKV0yv6WdHe5A=
</HostId>
</Error>
```

Figure 38: Output Access Denied the s3 mode

We found a link that gives us two links.

```
<!-- This XML file does not appear to have any style information associated with it. The document tree is shown below. -->
<?xml version="1.0"?>
<ListBucketResult>
  <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>2008-02-25T18:25:12.000Z</LastModified>
    <ETag>"d8f13cef769099ede1ce903bc289699a"</ETag>
    <Size>14352822</Size>
    <StorageClass>STANDARD</StorageClass>
  </Contents>
</ListBucketResult>
```

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.

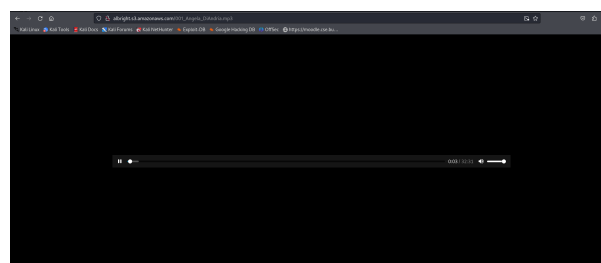


Figure 40: Output File for the s3 mode

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:

```
piyal@piyal-ROG-Strix-G531GT-G531GT:/tftpboot$ ls
abc.txt no.txt
```

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:

```
piyal@piyal-R0G-Strix-G531GT-G531GT:~/gobuster$ sudo cat tftpwords.txt
abc.txt
def.txt
haaa.txt
beee.txt
hahahaa.txt
wowowwww.txt
```

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.

```
piyal@piyal-R0G-Strix-G531GT-G531GT:~/gobuster$ ./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
=====
Found: abc.txt
Progress: 7 / 8 (87.50%)
=====
Finished
=====
```

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/setup-a-tftp-server-on-ubuntu-server/>.