

# Gotty 未授权访问漏洞分析报告

## 漏洞介绍

Gotty是一个开源的简单的命令行工具，基于go语言实现Linux终端Web共享。

Github: <https://github.com/yudai/gotty> Stars: 12681

现在很多公司的内网防火墙只允许http协议，从这样的网络去访问和控制云上的主机非常麻烦。GoTTY是一个用go语言开发的工具，它启动一个web应用服务，可以将任何指定的终端应用映射到指定的http端口，这样在防火墙内部的客户端就可以通过普通的浏览器Chrome, Firefox来访问。

## 影响范围

所有版本

## 漏洞简析

gotty中通过命令行参数 `--credential` 实现HTTP Basic认证，但是该方式默认是未开启的，如下图：

```
ubuntu@jiguang:/var/jiguang/terminal_file$ ./terminal_linux_amd64
NAME:
  gotty - Share your terminal as a web application

USAGE:
  gotty [options] <command> [<arguments...>]

VERSION:
  unknown_version+unknown_commit

OPTIONS:
  --address value, -a value      IP address to listen (default: "0.0.0.0") [$GOTTY_ADDRESS]
  --port value, -p value          Port number to listen (default: "8080") [$GOTTY_PORT]
  --permit-write, -w              Permit clients to write to the TTY (BE CAREFUL) [$GOTTY_PERMIT_WRITE]
  --credential value, -c value   Credential for Basic Authentication (ex: user:pass, [default disabled]) [$GOTTY_CREDENTIAL]
  --random-url, -r               Add a random string to the URL [$GOTTY_RANDOM_URL]
  --random-url-length value     Random URL length (default: 8) [$GOTTY_RANDOM_URL_LENGTH]
  --tls, -t                      Enable TLS/SSL [$GOTTY_TLS]
  --tls-crt value                TLS/SSL certificate file path (default: "~/.gotty.crt") [$GOTTY_TLS_CRT]
  --tls-key value                TLS/SSL key file path (default: "~/.gotty.key") [$GOTTY_TLS_KEY]
  --tls-ca-crt value             TLS/SSL CA certificate file for client certifications (default: "~/.gotty.ca.crt") [$GOTTY_TLS_CA_CRT]
  --index value                  Custom index.html file [$GOTTY_INDEX]
  --title-format value           Title format of browser window (default: "{{ .command }}@{{ .hostname }}") [$GOTTY_TITLE_FORMAT]
  --reconnect                    Enable reconnection [$GOTTY_RECONNECT]
  --reconnect-time value         Time to reconnect (default: 10) [$GOTTY_RECONNECT_TIME]
  --max-connection value         Maximum connection to gotty (default: 0) [$GOTTY_MAX_CONNECTION]
  --once                         Accept only one client and exit on disconnection [$GOTTY_ONCE]
  --timeout value                Timeout seconds for waiting a client(0 to disable) (default: 0) [$GOTTY_TIMEOUT]
  --permit-arguments             Permit clients to send command line arguments in URL (e.g. http://example.com:8080/?arg=AAA&arg=BBB) [$GOTTY_PERMIT_ARGUMENTS]
  --width value                  Static width of the screen, 0(default) means dynamically resize (default: 0) [$GOTTY_WIDTH]
  --height value                 Static height of the screen, 0(default) means dynamically resize (default: 0) [$GOTTY_HEIGHT]
  --ws-origin value               A regular expression that matches origin URLs to be accepted by WebSocket. No cross origin requests are acceptable
by default [$GOTTY_WS_ORIGIN]
  --term value                   Terminal name to use on the browser, one of xterm or hterm. (default: "xterm") [$GOTTY_TERM]
  --close-signal value           Signal sent to the command process when gotty close it (default: SIGHUP) (default: 1) [$GOTTY_CLOSE_SIGNAL]
  --close-timeout value          Time in seconds to force kill process after client is disconnected (default: -1) (default: -1) [$GOTTY_CLOSE_TIMEOUT]
  --config value                 Config file path (default: "~/.gotty") [$GOTTY_CONFIG]
  --version, -v                  print the version

Error: No command given.
ubuntu@jiguang:/var/jiguang/terminal_file$
```

通过 `main.go` 文件可知，`EnableBasicAuth` 选项取决于是否提供 `credential` 参数

```

▶ js
▶ pkg
└ resources
  favicon.png
  index.css
  index.html
  xterm_customize.css
▶ server
▶ utils
▶ vendor
▶ webtty
  .gitignore
  .gotty
  CONTRIBUTING.md
  go.mod
  go.sum
  help.go
  LICENSE
  main.go
/* Makefile
README.md
  screenshot.gif
version.go

```

```

56     exit(fmt.Errorf(msg), 1)
57 }
58
59 configFile := c.String("config")
60 _, err := os.Stat(homedir.Expand(configFile))
61 if configFile != "~/gotty" || os.IsNotExist(err) {
62     if err := utils.ApplyConfigFile(configFile, appOptions, backendOptions); err != nil {
63         exit(err, 2)
64     }
65 }
66
67 utils.ApplyFlags(cliFlags, flagMappings, c, appOptions, backendOptions)
68
69 appOptions.EnableBasicAuth = c.IsTrue("credential")
70 appOptions.EnableTLSClientAuth = c.IsTrue("tls-ca-crt")
71
72 err = appOptions.Validate()
73 if err != nil {
74     exit(err, 6)
75 }
76
77 args := c.Args()
78 factory, err := localcommand.NewFactory(args[0], args[1:], backendOptions)
79 if err != nil {
80     exit(err, 3)
81 }
82
83 hostname, _ := os.Hostname()

```

对HTTP Basic的认证在 `middleware.go` 文件的 `wrapBasicAuth` 函数中，而 `wrapBasicAuth` 函数在文件 `server.go` 中第199行被调用，取决于 `options.EnableBasicAuth` 的值，该值默认为 `false`，所以当启动时不添加 `--credential` 参数即会造成未授权访问

```

asset.go
handler_atomic.go
handlers.go
init_message.go
list_address.go
log_response_writer.go
middleware.go
options.go
run_option.go
server.go
slave.go
ws_wrapper.go
▶ utils
▶ vendor
▶ webtty
  .gitignore
  .gotty
  CONTRIBUTING.md
  go.mod
  go.sum
  help.go
  LICENSE
  main.go
/* Makefile
README.md
  screenshot.gif

```

```

21     w.Header().Set("Server", "GoTTY")
22     handler.ServeHTTP(w, r)
23 }
24
25 func (server *Server) wrapBasicAuth(handler http.Handler, credential string) http.Handler {
26     return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
27         token := strings.SplitN(r.Header.Get("Authorization"), " ", 2)
28
29         if len(token) != 2 || strings.ToLower(token[0]) != "basic" {
30             w.Header().Set("WWW-Authenticate", `Basic realm="GoTTY"`)
31             http.Error(w, "Bad Request", http.StatusUnauthorized)
32             return
33         }
34
35         payload, err := base64.StdEncoding.DecodeString(token[1])
36         if err != nil {
37             http.Error(w, "Internal Server Error", http.StatusInternalServerError)
38             return
39         }
40
41         if credential != string(payload) {
42             w.Header().Set("WWW-Authenticate", `Basic realm="GoTTY"`)
43             http.Error(w, "authorization failed", http.StatusUnauthorized)
44             return
45         }
46
47         log.Printf("Basic Authentication Succeeded: %s", r.RemoteAddr)
48         handler.ServeHTTP(w, r)
49     })
50 }
51
52

```

```

7 type Options struct {
8     Address      string      `hcl:"address" flagName:"address" flagSName:"a" flagDescribe:"IP
9     Port         string      `hcl:"port" flagName:"port" flagSName:"p" flagDescribe:"Port num
10    PermitWrite   bool       `hcl:"permit_write" flagName:"permit-write" flagSName:"w" flagDe
11    EnableBasicAuth bool      `hcl:"enable_basic_auth" default:"false"
12    Credential    string      `hcl:"credential" flagName:"credential" flagSName:"c" flagDescri
13    EnableRandomUrl bool      `hcl:"enable_random_url" flagName:"random-url" flagSName:"r" fla
14    RandomUrlLength int       `hcl:"random_url_length" flagName:"random-url-length" flagDescri
15    EnableTLS      bool      `hcl:"enable_tls" flagName:"tls" flagSName:"t" flagDescribe:"Ena
16    TLSCertFile   string      `hcl:"tls_crt_file" flagName:"tls-crt" flagDescribe:"TLS/SSL cer
17    TLSKeyFile    string      `hcl:"tls_key_file" flagName:"tls-key" flagDescribe:"TLS/SSL key
18    EnableTLSClientAuth bool      `hcl:"enable_tls_client_auth" default:"false"
19    TLSCACertFile string      `hcl:"tls_ca_crt_file" flagName:"tls-ca-crt" flagDescribe:"TLS/S
20    IndexFile     string      `hcl:"index_file" flagName:"index" flagDescribe:"Custom index.h

```

```
handler_atomic.go      195     siteMux.HandleFunc(pathPrefix+"config.js", server.handleConfig)
handlers.go           196
init_message.go       197     siteHandler := http.Handler(siteMux)
list_address.go       198
log_response_writer.go 199     if server.options.EnableBasicAuth {
middleware.go         200         log.Printf("Using Basic Authentication")
options.go            201         siteHandler = server.wrapBasicAuth(siteHandler, server.options.Credential)
run_option.go         202     }
server.go             203
slave.go              204     withGz := gziphandler.GzipHandler(server.wrapHeaders(siteHandler))
                           205     siteHandler = server.wrapLogger(withGz)
                           206
```

通过对静态资源分析可知，Gotty服务的 `index.html` 文件包含以下指纹信息，即成功登陆后服务器会返回 `"/js/gotty-bundle.js"` 字符串

```
<!doctype html>
<html>
  <head>
    <title>{{ .title }}</title>
    <link rel="icon" type="image/png" href="favicon.png">
    <link rel="stylesheet" href="./css/index.css" />
    <link rel="stylesheet" href="./css/xterm.css" />
    <link rel="stylesheet" href="./css/xterm_customize.css" />
  </head>
  <body>
    <div id="terminal"></div>
    <script src=".auth_token.js"></script>
    <script src=".config.js"></script>
    <script src=".js/gotty-bundle.js"></script>
  </body>
</html>
```

## 复现过程

### 复现环境

- Firefox
- Zoomeye.org

### 复现过程

#### 步骤1：

通过 `zoomeye.org` 搜索 `"/js/gotty-bundle.js"` 字符串，获得 `208` 个结果，分布还是挺广的

找到约 208 条结果 用时 0.158 秒

搜索结果 | 统计报告 | 全球视角 | 相关漏洞 | 分词 | 贡献 | 下载

./js/gotty-bundle.js

## 步骤2：

随机选择一个访问，可以成功执行命令

```
shell@Alicloud:~$ ls
link-iot-edge-x86-64-v1.8.4.tar.gz
shell@Alicloud:~$ id
uid=1000(shell) gid=1000(shell) groups=1000(shell)
shell@Alicloud:~$
```

## 痕迹分析

在 `middleware.go` 文件中函数 `wrapLogger` 实现了对访问者的记录，可根据该记录和历史命令判断攻击的访问痕迹

```
6
7
8
9    func (server *Server) wrapLogger(handler http.Handler) http.Handler {
10        return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
11            rw := &logResponseWriter{w, 200}
12            handler.ServeHTTP(rw, r)
13            log.Printf("%s %d %s %s", r.RemoteAddr, rw.status, r.Method, r.URL.Path)
14        })
15    }
16 }
17
18 func (server *Server) wrapHeaders(handler http.Handler) http.Handler {
19     return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
20         // todo add version
21         w.Header().Set("Server", "GoTTY")
22         handler.ServeHTTP(w, r)
23     })
24 }
```

## 防护方案

启动时通过命令行参数 `--credential` 实现HTTP Basic认证

## 参考链接

[1] Gotty

<https://github.com/yudai/gotty>

[2] Zoomeye.org

<https://zoomeye.org>

极光@知道创宇