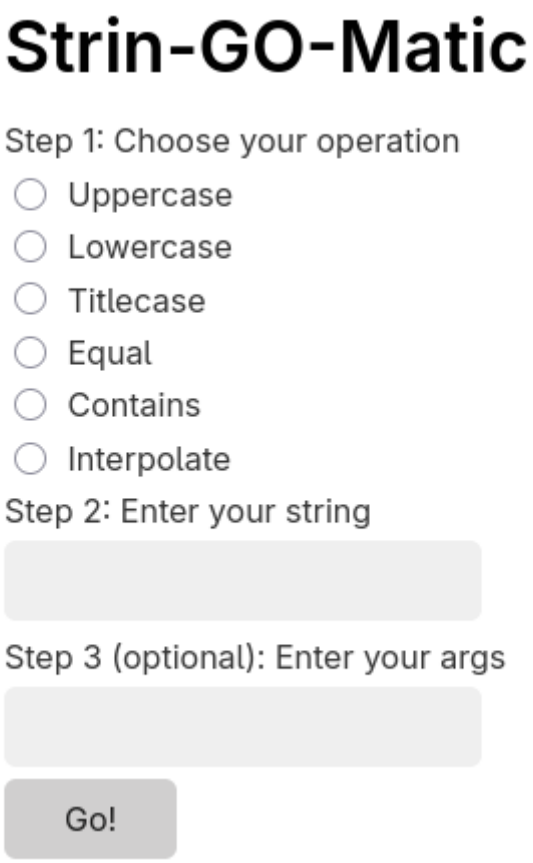
Strin-GO-Matic

Written by `@mxg586` (Misha)

This is a misc challenge which has a solution which is not like any challenge I have seen before.

The challenge uses a web-app written in Go which allows you to enter one or two strings and performs an operation on those strings.



There are six operations you can perform on the strings which change the case, compare two strings and other similar operations. The 'args' box is only used by the 'Equal', and 'Contains' operations since they involve two strings instead of one.

When you look at the `main.go` file, which contains the back-end code, you can see that only one operation can be made to print the flag, that being the 'Interpolate' operation. Here is the code for that operation:

```
case "interpolate":
    words := strings.Split(command.Input, " ")
    for i, word := range words {
```

```
        if envVarPattern.MatchString(word) {
            varName := lower.String(word[1:])
            for _, substring := range forbiddenSubstrings {
                if start, end := matcher.IndexString(varName, substring); start != -1 && end
!= -1 {
                    invalid(w, http.StatusForbidden, "Forbidden variable")
                    return
                }
            }

            varValue, ok := os.LookupEnv(word[1:])
            if !ok {
                varValue = "{VARIABLE NOT SET}"
            }

            words[i] = varValue
        }
    }
    result = strings.Join(words, " ")
```

In summary, the above code uses a regex pattern to look for 'words' in the input. It loops over all strings split by spaces and checks if the lowercase version of any of them matches any forbidden strings. If it is a forbidden string then it tells you that it is a forbidden string. Otherwise it outputs your string but with the 'words' replaced with their corresponding environment variables (if such variables exist).

For example, an input of `before $PATH after` gives the output `before /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin after`.

If you give it the input `$STRIN_GO_MATIC_FLAG`, which contains the flag, then it says `Forbidden variable`, since `strin_go_matic` is a forbidden substring.

My first thought on how to solve this challenge was to check if there is any vulnerability in the regex string the program used:

```
^\$[A-Za-z0-9_]+$
```

However, after putting the string into an online regex explainer I could not see any issues. The regex seems to do exactly what it is supposed to do, and won't allow you to put variables in the middle of other variables. Things like `$PATH$STRIN_GO_MATIC_FLAG` didn't work.

After failing to find a way around the regex matching, I spent a while just looking at `main.go` trying to find another way to solve the challenge. Most of rest of the code, apart from putting the flag in

an environment variable, is just regular web server stuff that didn't seem to be part of the challenge.

However, there was one part of the code which stuck out to me as odd:

```
var tag language.Tag

acceptedLocales := strings.Split(r.Header.Get("Accept-Language"), ",")
for _, option := range acceptedLocales {
    if strings.Contains(option, ";") {
        option = strings.Split(option, ";")[0]
    }

    var err error
    tag, err = language.Parse(option)
    if err == nil {
        continue
    }
}

if tag == (language.Tag{}) {
    tag = language.AmericanEnglish
}

upper := cases.Upper(tag)
lower := cases.Lower(tag)
title := cases.Title(tag)

matcher := search.New(tag)
```

This code checks the HTTP headers for a language tag, defaulting to American English if none was specified. It then uses the tag to set the behaviour of the uppercase, lowercase and title case functions.

If this were an actual web-app, I probably wouldn't have thought much of this code, but for a CTF challenge it feels weird that the challenge checks the language when it doesn't seem to have much of an impact on the webpage.

Then, I looked at the 'Interpolate' code again and realised that the code checks the lowercase version of your input against forbidden strings but uses the original version of your input for retrieving the variables. I then figured that I might be able to get the flag if I somehow made the lowercase version of `STRIN_GO_MATIC_FLAG` not contain the string `strin_go_matic`.

I then started looking at different countries' alphabets and writing systems to see if there were any which had uppercase Latin letters that were not standard Latin letters in their lowercase form. I tried Cyrillic which didn't work since all the letters are non-Latin (even the ones that look the same as Latin letters).

Then I looked at the Turkish alphabet which looked promising. Some of the letters looked identical to Latin ones and others didn't. I then found that Turkish has the capital letter `I` whose lowercase is `ı`. I pasted both of them into a special character finder which finds letters not on the US keyboard and sure enough, the uppercase letter is Latin and the lowercase one isn't.

This means that when I enter `$STRIN_GO_MATIC_FLAG`, the lowercase version will almost match `strin_go_matic` but the `I`s will be the Turkish lowercase `ı` so it won't match the forbidden substring.

So, I typed `$STRIN_GO_MATIC_FLAG` into the string box, pressed 'Go!' and then copied the unsuccessful request as a cURL command from the network tab of devtools. Here is the request after I have removed all the unnecessary headers:

```
curl 'http://localhost:9000/go' \
  -X POST \
  -H 'Accept-Language: en-US,en;q=0.5' \
  -H 'Content-Type: application/json' \
  --data-raw '{"operation":"interpolate","input":"$STRIN_GO_MATIC_FLAG","args":""}'
```

To get the flag, I just changed the `Accept-Language` header to be Turkish:

```
curl 'http://localhost:9000/go' \
  -X POST \
  -H 'Accept-Language: tr' \
  -H 'Content-Type: application/json' \
  --data-raw '{"operation":"interpolate","input":"$STRIN_GO_MATIC_FLAG","args":""}'
```

After I ran this on the remote server instead of `localhost`, I got the flag! The flag contained something along the lines of `how_many_I_can_there_be` so I could tell I had probably found the intended solution (and didn't even need to recompile `libc`!)

Just to check, I added a print statement to the `main.go` file and as I expected, the program converted my input to `strın_go_matıc_flag` where it matches `strin_go_matic` in every letter except the `I`s.

The funny thing is that this challenge can also be solved by changing your browser's language to Turkish so if a Turkish speaker tried this challenge, it is likely the challenge would just give them the flag without them having to do anything!

The solution for this challenge wasn't complicated, but figuring it out still took a while. I really enjoyed this challenge due to the unique solution.

Revision #1
Created 2025-12-09 01:10:59 UTC by AFNOM
Updated 2025-12-09 01:12:33 UTC by AFNOM