Platypwn 2025

A collection of writeups from https://ctftime.org/event/2606.

Mirror of the challenges available here: https://github.com/sajjadium/ctf-archives/tree/main/ctfs/Platypwn/2025

Men At Work

Written by `@jb007db5`.

Men At Work was a quite difficult OSINT challenge that was a great example of using multiple people's knowledge and observations to form a solve. The correct solution required dropping a pin on a map for where the photo was taken and specifying the date

Some basic observations were easy to make. I could tell by experience that the station looked Scandinavian, so was likely either in Sweden, Finland, Norway or possibly Estonia or Latvia. Reverse image search quickly confirmed this suspicion, returning a handful of different stations. Very quickly, someone else in our team was able to narrow it down to the correct station, Turku Åbo, aka Turku Central Station.

To narrow down the exact location, we noted the relative position of the buildings around the bridge, correctly identifying the one it was taken from. I then worked out the exact spot on the bridge by observing the track layout below, working out that it was just to the left of one of the lifts on the bridge. This was clearly the easy part of the challenge, as finding the correct date would prove far more difficult.

I saw that the tracks were blocked by work vehicles, likely meaning the station would have been closed when the picture was taken. A quick google identified that the station had been undergoing major work since 2022, and the relative complete state in the photo limited it to 2024 or 2025. There was an outdoor concert style stage visible in the image so I found station closure dates and tried to line them up with concert dates that another team member had found, but none lined up neatly. They also discovered that The Voice Finland films in a building adjacent to Turku station, but this was once again a dead end, as they do not use outdoor stages. By increasing image

contrast, we could make out some text on the stage, that I believed spelt '_aü', with the first letter hidden. A small marquee to the right had 'ha' and then a letter with an umlaut, so we tried to find any concerts by a band with that name or what it might mean in Finnish but found no useful results. We then took a break from the challenge after running out of ideas.

Another of member of our team returned to the challenge many hours later and was able to increase contrast even further. Their work made it clear that the word on the stage was actually '_aj'. I spotted that the first letter looked like it might be a K and they identified that it could be the Finnish band Kaj, who represented Sweden in the 2025 Eurovision Song Contest, due to their lyrics primarily being in Swedish. A quick google found that they had performed in Turku on August 20th 2025. Inputting this date along with our pindrop gave us the correct flag and the challenge was thus complete.

Answer:

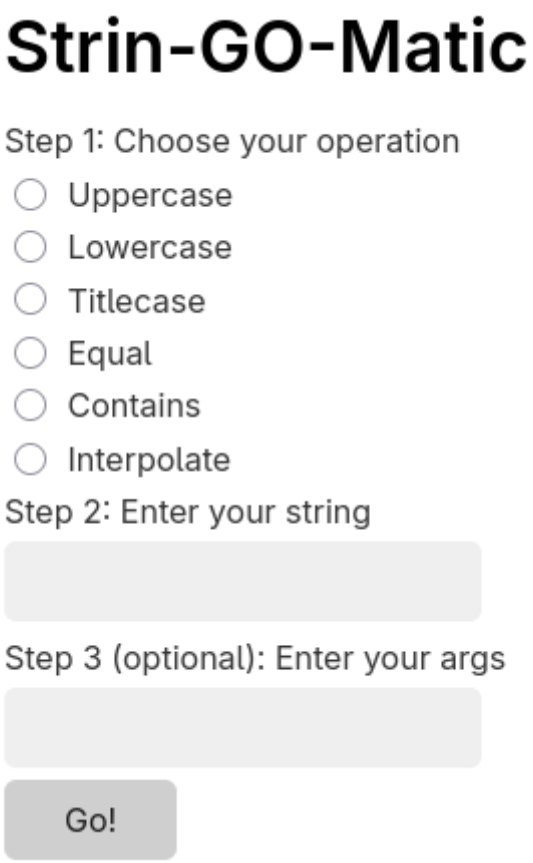
- Location: Approx. 60°27'22.5"N 22°15'32.0"E
- Date: 20th August 2025

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.