Challenge Sticker

Write-up by Valentin BOUSSON

Intro

Ce premier challenge était simplement nommé « Challenge Sticker », dans la section Misc, et sa résolution valait 50 points. Un challenge plutôt simple dans la compétition.

Sans indice, le titre indiquait surtout qu'un code était présent sur le sur le sticker distribué à l'entrée de la conférence. En effet, l'ASCII art de la mascotte semblait déjà louche : beaucoup de symboles inséré dans le logos semblent représenter une grosse expression régulière.

Après plusieurs tentatives d'OCR infructueuses (la police étant réduite et le texte non human-friendly), j'ai pris mon courage à 2 mains pour retaper cette regex, en évitant tous les caractères '&'. La voici en entier :

```
^(?=\A|^)GH19(?P<New>^\?|{)\d{1}(?<=[(^1)|\A0+])\w(?<![^h])\?(?=C\w[^1-9].\
b).{1}m\dn\.{2}\b(?=\w)(?:)(?P<isnot>(?=\w*k\b).(?<=G)r(?!:\\)(?=\wH)e\B.[a
-c&^b]{2}(?<=[A-Z]a.).)(?<=[c-e]k)(?!:\1|always)=\\{0}\ds(?<!0.)(?<=1\D).\d
+(?<=\.4{4})w3s(?:[^!|e]|\De)(?<=m.)!(?P<better><3}\Z)$
```

L'épreuve peut démarrer !

Ressources

Documentation qui a aidé à comprendre les patterns spéciaux:

- https://www.rexegg.com/regex-quickstart.html
- https://www.regular-expressions.info/named.html

Outils de matching et d'explication de regex -> super intéressants :

- https://regex101.com/
- https://regexr.com/4pokt
- https://www.debuggex.com/

Pour en apprendre plus sur les regex, ou s'entraîner à les manipuler :

- https://regexcrossword.com/

Résolution / Soluce - / ! \ Attention spoiler !

Attention, La page suivante décrit la façon dont j'ai résolu cette énigme. Si vous voulez la résoudre vous-même, arrêtez la lecture ici, vous avez déjà suffisamment d'info ! Sinon, suivez-moi !

Alors, en sachant à l'avance que tous les flags de la compétition valideraient déjà l'expression **^GH19{.*}$** (c'est le cas dans la plupart des CTF, ça facilite la reconnaissance des flags), on pouvait facilement supposer que cette nouvelle regex admettrait une solution unique, le flag.

Pour une expression régulière simple, on aurait pu trouver des solveurs automatiques, mais la complexité de cette regex Python (ou PCRE) vient de l'utilisation intensive de back-reference, des forward-reference, et des capturing groups. Alors faisons-le à la main, cela fera réviser ces différents outils :

`^(?=\A|^)GH19…`
On se doutait donc qu'en début de chaîne, ce pattern n'allait ne rien matcher d'autre que **GH19**, mais pour la compréhension, cette syntaxe exprime un positive-lookahead, qui ne consomme pas de caractère, mais qui s'assure que la suite de la chaîne testée match le pattern décrit, ici un début de string ou un début de ligne. Pas franchement intéressant de tester ces méta-caractères…
On remplacera donc ce pattern par **^GH19**

`…19(?P<New>^\?|{)…`
Cette pattern décrit un *named captured group*, c'est-à-dire qu'il match le pattern **^\?|{**, et nomme cette chaîne reconnue « New ». Cela aurait eu un intérêt si plus loin, un appel à **\k<New>** avait été décrit, pour y faire référence, mais ce n'est pas le cas, ce n'est que de l'obfuscation. Le pattern reste néanmoins **(^\ ?|{)**, et on observe facilement que la première branche du choix n'est pas possible puisque cela nécessiterait de commencer par un début de ligne. On remplace donc tout ce groupe par le simple **{**, qui produit comme on s'y attendait au préfixe **GH19{**. Continuons.

`…\d{1}(?<=[(^1)|\A0+])…`
Pas beaucoup plus dur, mais cette fois-ci avec le concept de *positive-lookbehind*, qui vérifie que les caractères précédent ce pattern le matchent bien. On a donc un chiffre (**\d{1}**), qui doit faire partie de cette liste de caractères [(^1)|\A0+]. Notez que c'est une écriture fourbe, puisque il n'y a pas de OU (**|**), ni même de négation (**^**), ni de 0 répété (**0+**) dans ce format d'expression régulière ; c'est donc une simple liste de caractère (syntaxe **[abc]**). Or, nous savons que nous avons un chiffre avant, nous pouvons tout simplement limiter le pattern à **[01]**.

`…\w(?<![^h])…`
Ici, un *negative-lookbehind*. On vérifie que le caractère précèdent ne match pas le pattern **[^h]**, c'est donc une double-négation et par conséquent la simple lettre **h**.

`…(?=C\w[^1-9].\b).{1}m\dn\.{2}\b…`
- ⇨ **Cm0n\.\.\b**

`…(?=\w)(?:)(?P<isnot>(?=\w*k\b).(?<=G)r(?!:\\)(?=\wH)e\B.[a-c&^b]{2}(?<=[A-Z]a.).)(?<=[c-e]k)…`
- ⇨ (?=\w)(?=\w*k\b).(?<=G)r(?!:\\)(?=\wH)e\B.[a-c&^b]{2}(?<=[A-Z]a.).(?<=[c-e]k)
- ⇨ (?=\w*k\b)Gr(?!:\\)(?=\wH)e\B.[a-c&^b]{2}(?<=[A-Z]a[c-e])k
- ⇨ (?=\w*k\b)Gr(?!:\\)eHa[a-c&^b]k
- ⇨ **GreHack\b**

`…(?!:\1|always)=\\{0}\ds(?<!0.) (?<=1\D)…`
\1 fait reference au tout premier groupe de la string, celui contenant des débuts de ligne/de string,il n'est donc pas possible. **always** n'est pas possible non plus. **\\{0}** correspond à 0 occurrence du caractère ****. Inutile également.
- ⇨ **=1s**

`….\d+(?<=\.4{4})…`
- ⇨ **\.4444**

`…w3s(?:[^!|e]|\De)(?<=m.)!…`
- ⇨ **w3sme!**

`…(?P<better><3}\Z)$`
- ⇨ **<3}$**

Tout ça mit bout à bout nous donne une regex beaucoup plus claire:
^GH19{[01]h\?Cm0n\.\.GreHack=1s\.4444w3sme!<3}$

Qui admet de façon surprenante 2 solutions:
- *GH19{0h?Cm0n..GreHack=1s.4444w3sme!<3}*
- *GH19{1h?Cm0n..GreHack=1s.4444w3sme!<3}*

Avant de les donner aux organisateurs, et pour s'assurer du résultat, on valide avec un outil classique qu'elles matchent toutes les 2 la regex donnée en consigne, ce qui est le cas.

Bingo, 2 solutions pour le prix d'une !