Pre-Chall

Auteur: bWlrYQ

1. Etape 1

Nous n'avons pas vraiment d'information conçernant le point d'entrée, on peut donc fouiller les réseaux sociaux (rien) ainsi que tester le bot Discord `[PréChall] FlagMalo#5729` mais ça ne donne rien. Il nous reste le site du CTF Flag'Malo.

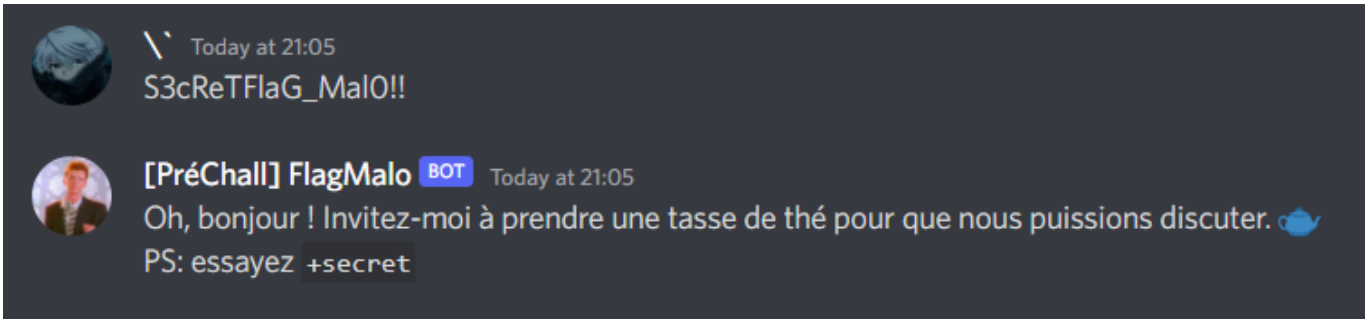
Dans le code source de la page par défaut on peut trouver en bas cette chaîne de caractères en commentaire:

```
<!--
4gFVUr7qAa2F9GNCfcxf6rwWbKLwqPHASRDNYJzGePfxsdyoT2XR5GLrcPVeyU68mguNHsVHkSH8t45rBV
fkJh3NqDUhxFkyZbEiVq5Rte1a1T2o7fdav1DfrVg9vbyAzCXEmRdebGN8ANBAAAS3t2fm -->
```

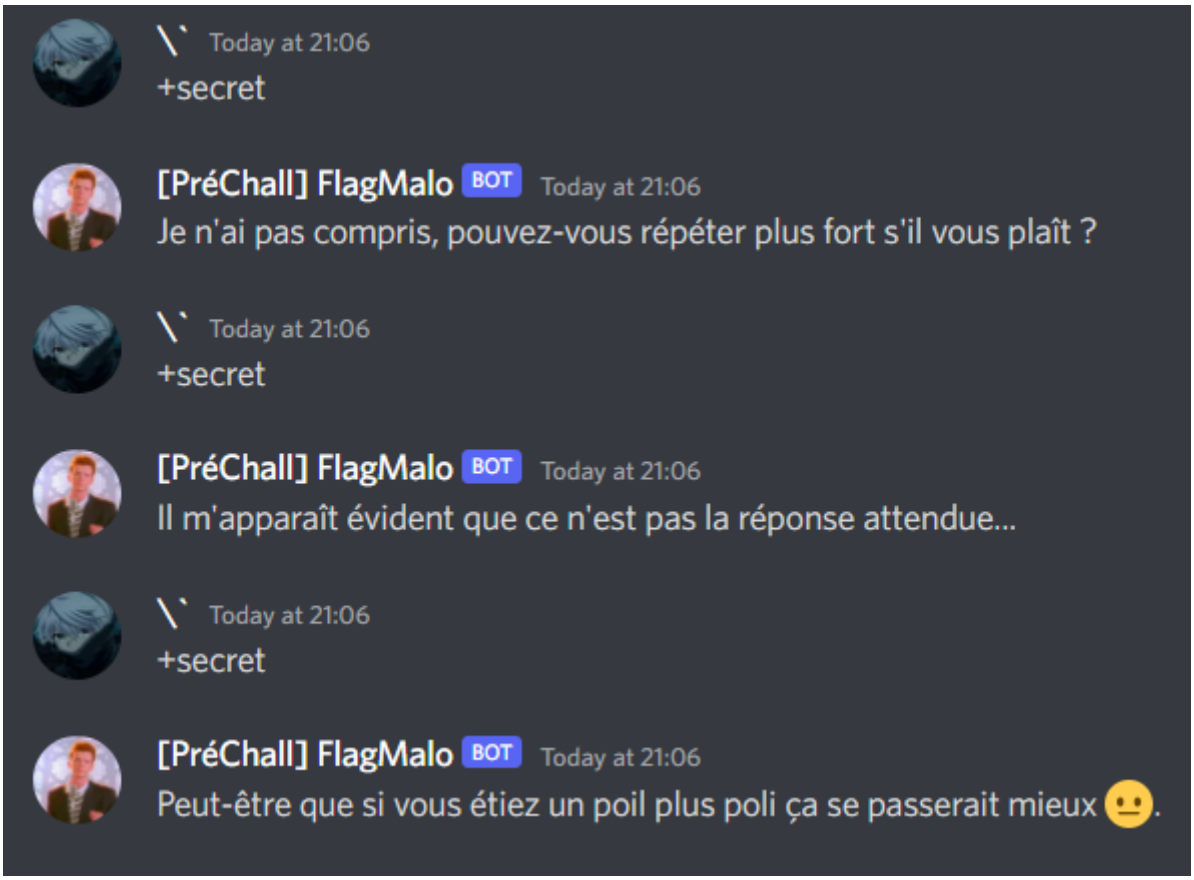
Elle semble encodée, on peut tester différents encodages, ici c'est de la base 58, on peut la décoder à l'aide de CyberChef. On en sort la phrase suivante: `Vous pouvez envoyer le mot de passe 'S3cReTFlaG_Mal0!!' en message privé à [PréChall] FlagMalo#5729 sur Discord`. On passe donc à l'étape 2.

2. Etape 2

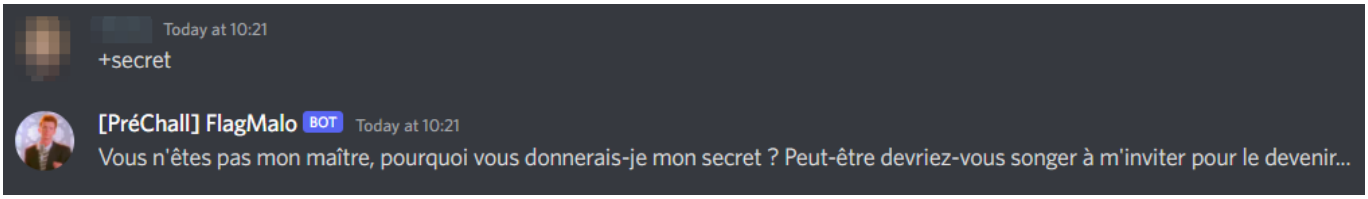
On envoie donc `S3cReTFlaG_Mal0!!` au bot qui nous répond la chose suivante:



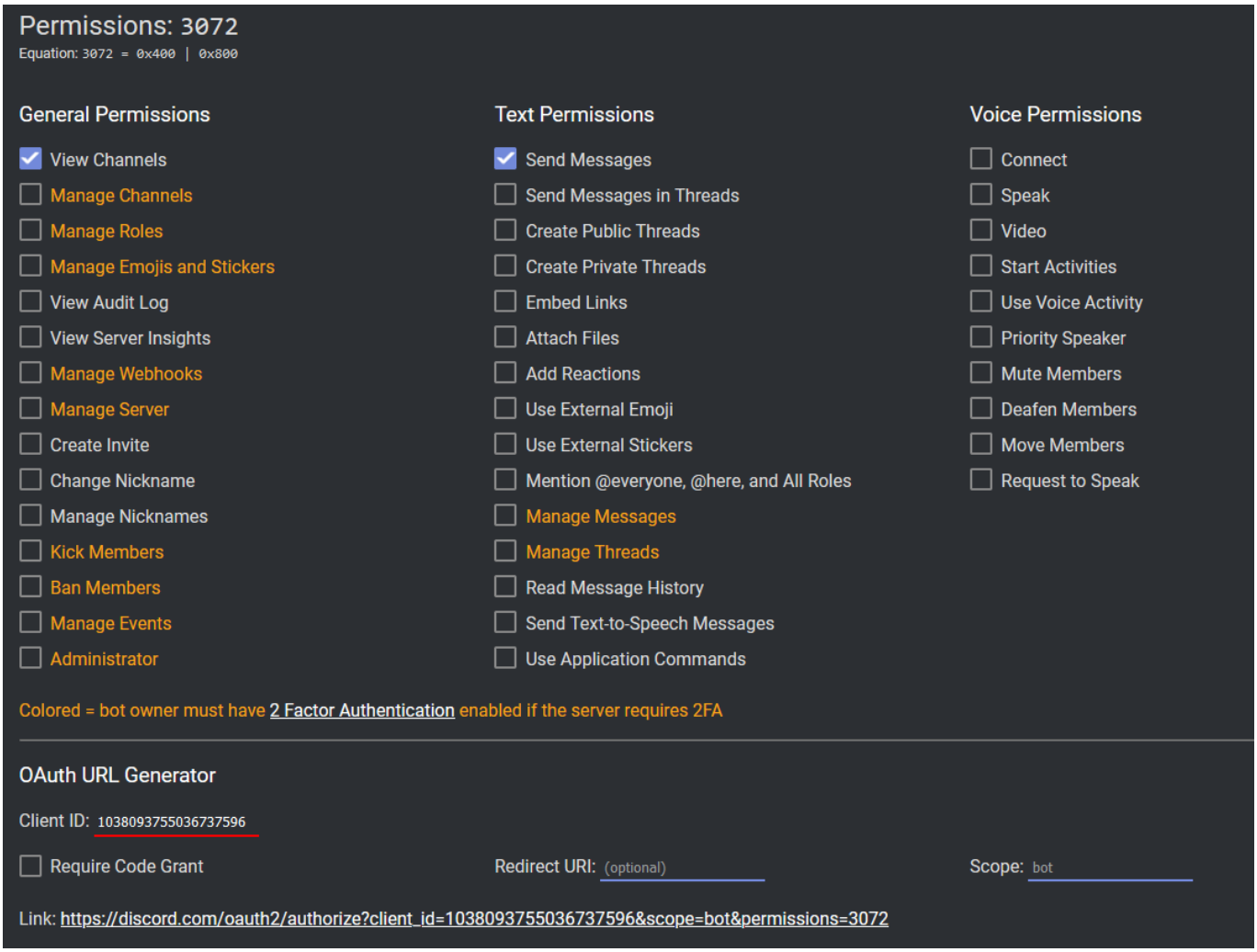
On essaie `+secret`



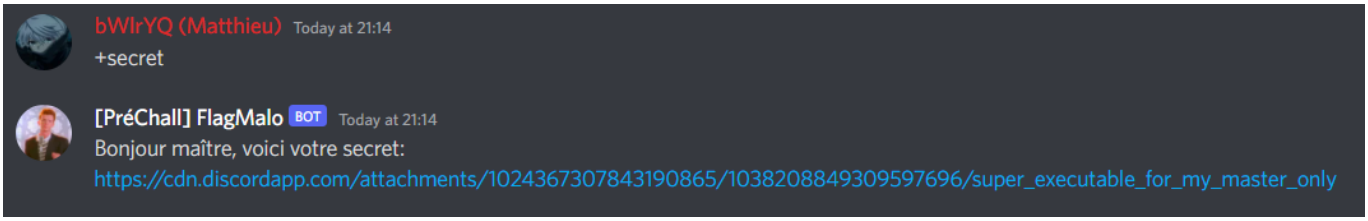
Cela ne donne rien, on peut essayer toute une combinaison de choses mais rien ne fonctionne. En revanche rien ne nous force à rester en DM avec le bot on peut donc l'utiliser sur le serveur FlagMalo dans le canal d'équipe:



Le résultat est cette fois-ci différent, si on fait un parallèle entre le message au moment où on donne le mot de passe et maintenant on comprend qu'il faut l'inviter (et être son maître) pour qu'il puisse nous donner son secret. Par défaut les bots sont "publics" sur Discord, c'est à dire que n'importe qui peut les inviter à être utilisés sur un serveur qui nous appartient. On peut utiliser ce site qui permet de générer un lien d'invitation basé sur l'id du bot.



Il ne nous reste plus qu'à inviter le bot sur un Discord qui nous appartient et à faire `+secret`



3. Etape 3

Si on se rend sur le lien donné par le BOT on peut télécharger un fichier sans extension. Pour identifier aisément son contenu on peut utiliser la commande `file`

```
┌─mika at bwlryq in ~
└─λ file super_executable_for_my_master_only                                    0 (0.005s) < 20:15:36
super_executable_for_my_master_only: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interp
reter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=e8a34073fea4ecc770ea738c061612b04173d619, for GNU/Linux 3.2.0, not stri
pped
```

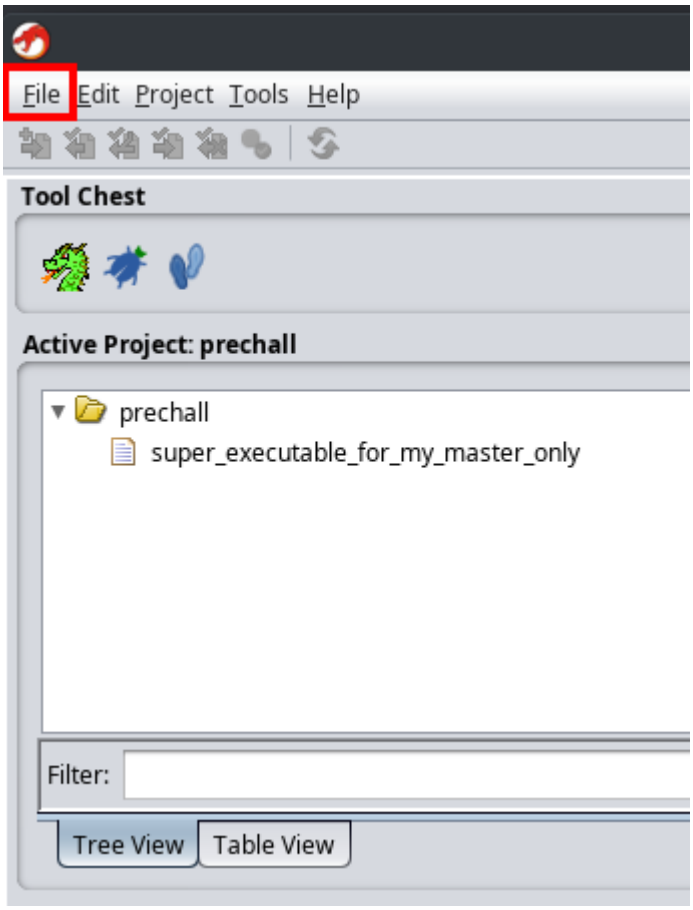
C'est un fichier ELF donc un exécutable compilé pour les sytèmes UNIX on peut donc l'exécuter.

```
┌─mika at bwlryq in ~
└─λ chmod +x super_executable_for_my_master_only                                0 (0.015s) < 20:15:38
┌─mika at bwlryq in ~
└─λ ./super_executable_for_my_master_only                                       0 (0.004s) < 20:17:03
Maître, quel est votre code à 4 chiffres ?AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Raté, faites un effort...
┌─mika at bwlryq in ~
└─λ                                                                             0 (4.529s) < 20:17:10
```

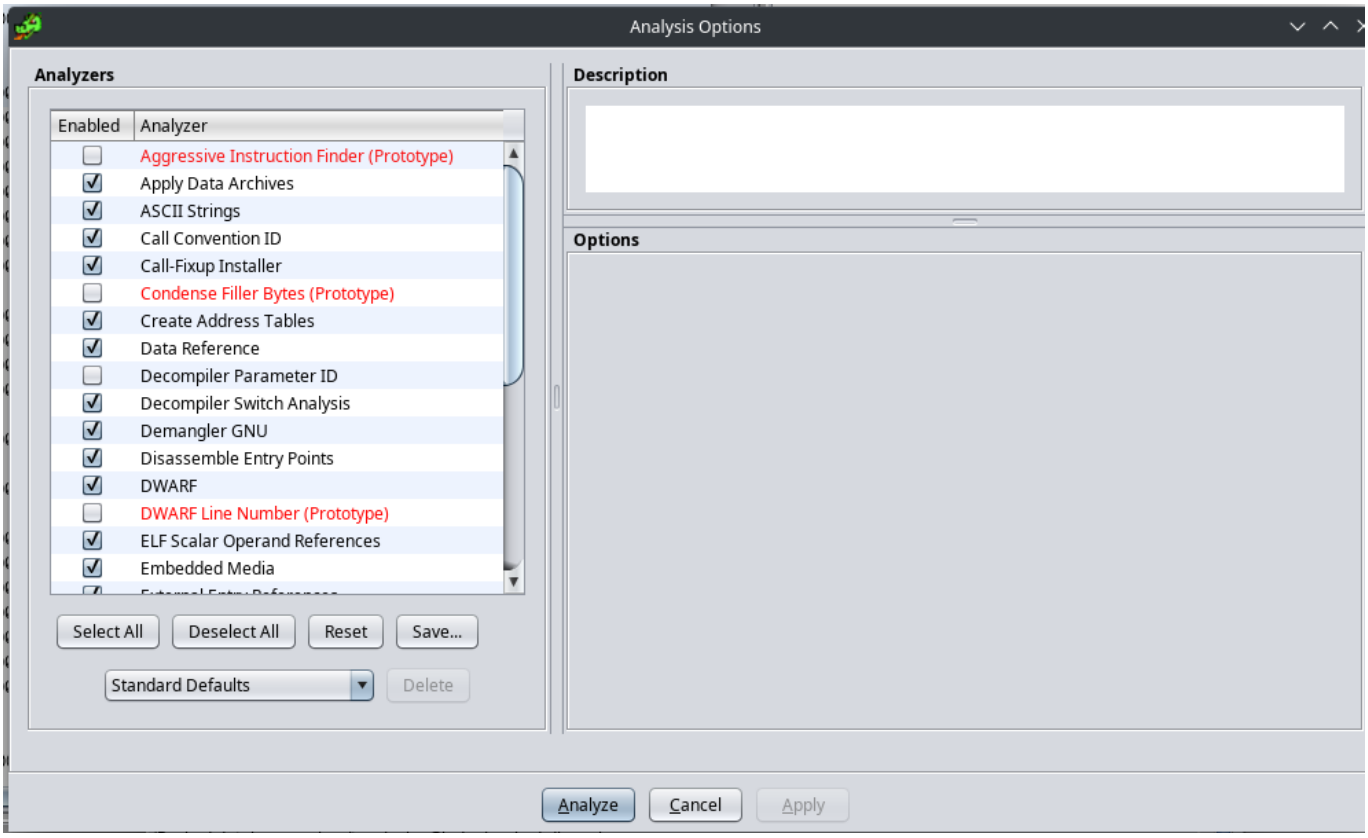
Il faut donc trouver un code à 4 chiffres pour découvrir les secrets du binaire. Nous avons deux possibilités, bruteforce le code (grâce à l'information qui nous est donnée) soit faire un peu de rétro-ingénierie sur le programme. Pour le bruteforce (pas marrant + nul), on peut faire cela (credits pour ZUMBA#1916):

```bash
#!/bin/bash
for i in {0001..9999}
do
    yes "$i" | ./super_executable_for_my_master_only
    echo "$i"
done
```

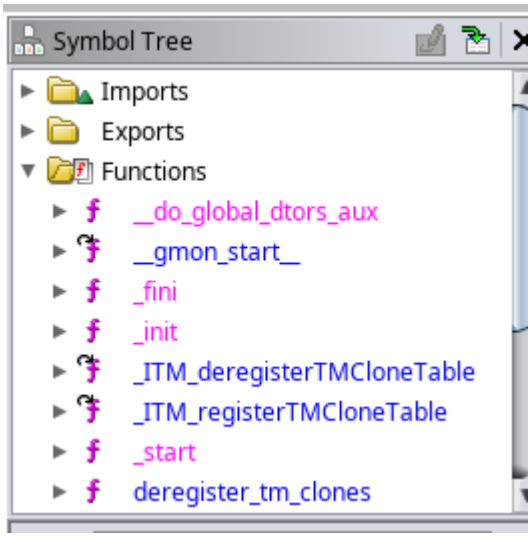
Pour la partie intéressante conçernant la rétro-ingénierie notons 3 choses, premièrement on nous affiche un message, on nous demande une entrée utilisateur qui est censée être un entier à 4 chiffres. enfin ,on nous affiche un message dépendant de ce que l'on a entré. Pour l'analyse nous utiliserons Ghidra.
On créé un nouveau projet, on importe le binaire



On lance Ghidra sur le binaire et l'analyse



Dans l'arbre des symboles on peut voir les différentes fonctions, intéressons nous à la fonction main()



En regardant le code décompilé on peut voir la fonction décompilée

Decompile: main - (super_executable_for_my_master_only)

```
1
2  undefined8 main(void)
3
4  {
5    long in_FS_OFFSET;
6    int local_14;
7    long local_10;
8
9    local_10 = *(long *)(in_FS_OFFSET + 0x28);
10   printf(&DAT_00102008);
11   scanf("%u",&local_14);
12   if (local_14 == 0xa33) {
13     printf(&DAT_00102030);
14   }
15   else {
16     printf(&DAT_0010208b);
17   }
18   if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
19                     /* WARNING: Subroutine does not return */
20     __stack_chk_fail();
21   }
22   return 0;
23 }
24
```

On peut voir la déclaration d'un entier local_14, un printf (affichage terminal en C), un scanf (entrée utilisateur), une condition if qui vérifie la valeur de local_14 en la comparant avec 0xA33 (2611 en base 10). On comprend alors que si local_14 = 2611 alors on a un affichage, dans le cas contraire on en a un autre. Notre code est donc 2611. On peut confirmer en exécutant le binaire:

```
mika at bwlryq in ~
λ ./super_executable_for_my_master_only                    0 (4.529s) < 20:17:10
Maître, quel est votre code à 4 chiffres ?2611
Vous pouvez donner ce code à un administrateur du FlagMalo, il vous donnera votre flag :)
```

On envoie ce code à un admin et il nous donne le flag: `FMCTF{PreChallenge_Is_T00_E@sY_F0r_You}`.