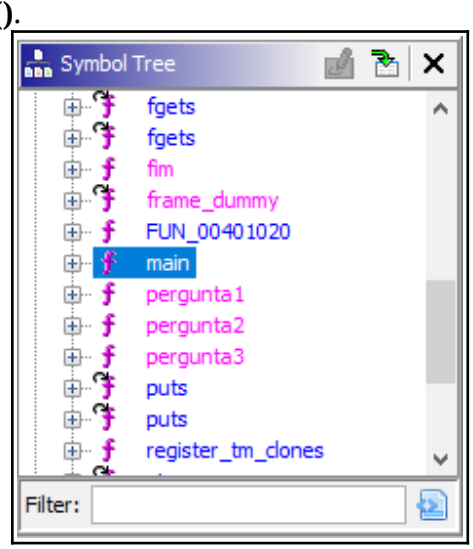
Aluno: José Victor Medeiros Thomé da Silva
Tag de Engenharia Reversa

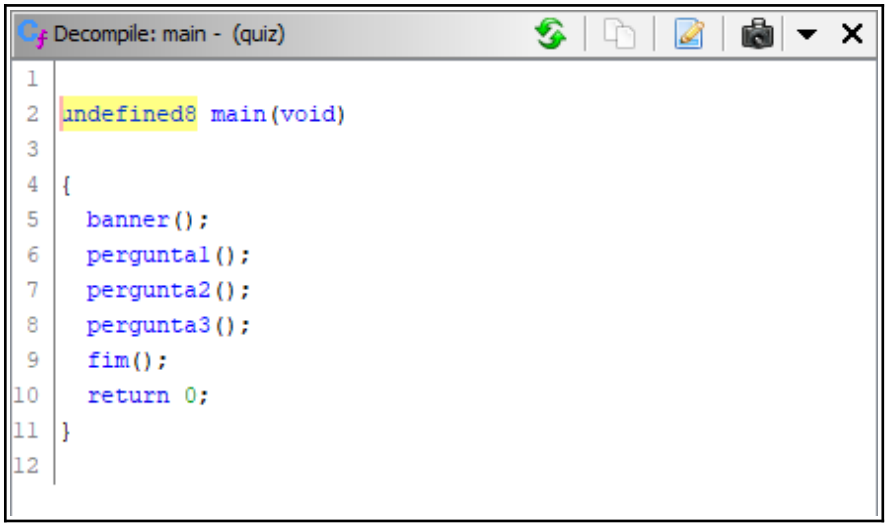
Writeup

- **Primeiros passos:**

  Para analisar o arquivo “quis” eu usei o sistema operacional Windows e o programa de engenharia reversa Ghidra, e para executar o código eu usei o compilador gcc instalado num WSL do Ubuntu.
  Para localizar cada função eu usei a Symbol Tree do Ghidra, me orientando através da função **main()**.



  Para analisar cada código eu me vali do descompilador.



- **Pergunta 1:**

  Eu consegui descobrir a resposta correta dessa pergunta basicamente analisando o código no descompilador. Nele percebi no “puts” da linha 11 a pergunta feita, e a linha 13 entregou claramente que a resposta era “cache,obvio” por estar executando uma função de comparação (strcmp) entre “cache,obvio\n” e o array de caracteres que recebe a entrada do usuário na linha 11, o **local_58**. O que me fez perceber que **local_58** era o array que recebia a entrada do usuário passada no terminal foi o fato de o mesmo ser parâmetro de entrada no “fgets” executado na linha 12.

```
2  void perguntal(void)
3
4  {
5    int iVar1;
6    long in_FS_OFFSET;
7    char local_58 [72];
8    long local_10;
9
10   local_10 = *(long *)(in_FS_OFFSET + 0x28);
11   puts("1) Quem eh mais rapido, ram ou cache?");
12   fgets(local_58,0x3f,stdin);
13   iVar1 = strcmp(local_58,"cache,obvio\n");
14   if (iVar1 == 0) {
15     puts("Resposta correta!");
16   }
17   else {
18     ganhou = 0;
19   }
20   if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
21                     /* WARNING: Subroutine does not return
22     __stack_chk_fail();
23   }
24   return;
25 }
```

- **Pergunta 2:**

  Nesta pergunta eu precisei usar uma tabela **ASCII** para conseguir me guiar. Eu vi que na linha 27 do código descompilado estava fazendo uma comparação com o número hexadecimal 0x41, que na tabela ASCII representa a letra ‘A’. Vendo que o valor da variável “local_88” estava sendo usado para fazer uma operação de XOR bit a bit com a entrada, decidi na minha implementação imprimir seu valor no formato string para saber do que se tratava:

```
30      puts("2) o que o rip faz?");
31      fgets((char *)local_58,0x3f,stdin);
32      sVar1 = strlen((char *)&local_88);
33      printf("%s\n", (char *)&local_88);
```

  Daí eu vi que o valor era “ .2”, isto é, dois espaços, um ponto e o número dois. O interessante é que no if de comparação desse código, o teste é feito caractere por caractere tanto de “local_88” quanto de “local_58” que armazena a entrada do usuário.

```
if ((*(char *)((long)&local_88 + (long)local_90) ^ local_58[local_90]) != 0x41)
```

  Diante desse fato, acabei descobrindo que nessa operação do if o XOR bit a bit executado entre o valor de entrada e o valor de “local_88” é realizado em um caractere por loop, e como os dois primeiros valores da string em “local_88” são espaços, os dois primeiros caracteres da string passada pelo usuário são usados para fazer XOR com o número 32 (o formato decimal do caractere espaço na tabela ASCII), o que coincidentemente é uma operação que pode ser usada para transformar caracteres letras minúsculas em maiúsculas e vice versa.

  Logo, para acertar essa questão, basta o usuário digitar a letra ‘a’ minúscula ou escrever qualquer palavra ou frase na qual a letra ‘a’ minúscula apareça como primeiro ou segundo caractere.

Decompile: pergunta2 - (quiz)

```
2  void pergunta2(void)
3
4  {
5    size_t sVar1;
6    long in_FS_OFFSET;
7    int local_90;
8    undefined8 local_88;
9    undefined8 local_80;
10   undefined8 local_78;
11   undefined8 local_70;
12   undefined local_68;
13   byte local_58 [72];
14   long local_10;
15
16   local_10 = *(long *)(in_FS_OFFSET + 0x28);
17   local_88 = 0x316120352f2e3120;
18   local_80 = 0x3331612061203320;
19   local_78 = 0x2f2861202c28392e;
20   local_70 = 0x4b2e202234333532;
21   local_68 = 0;
22   puts("2) o que o rip faz?");
23   fgets((char *)local_58,0x3f,stdin);
24   sVar1 = strlen((char *)&local_88);
25   local_90 = 0;
26   while (local_90 < (int)sVar1) {
27     if ((*(byte *)((long)&local_88 + (long)local_90) ^ local_58[local_90]) != 0x
28       ganhou = 0;
29     }
30     local_90 = local_90 + 1;
31   }
32   puts("Resposta correta!");
```

- **Pergunta 3:**

  Observando o código dessa pergunta descompilado no Ghidra, deu para notar de cara que era uma comparação de um valor específico em hexadecimal com um valor dado pelo usuário, porém teve um valor “&DAT_0040208d” dentro de uma instrução “__isoc99_scanf” no qual eu não entendia de fato. Fazendo uma rápida pesquisa na internet, descobri que “__isoc99_scanf” na verdade se trata da função “scanf” da versão C99 da linguagem C. Diante desse fato, sabendo que o primeiro parâmetro do “scanf” é o formato que se deseja converter a entrada passada pelo usuário, procurei no Assembly qual string de formato “&DAT_0040208d” estava armazenando, então encontrei essa parte:

```
                    DAT_0040208d
0040208d 25             ??          25h    %
0040208e 6c             ??          6Ch    l
0040208f 75             ??          75h    u
```

  Daí descobri que “&DAT_0040208d” representava “%lu”, ou seja, o scanf estava convertendo a entrada do usuário para “unsigned long int”. Com essa informação em maos, peguei o código obtido na engenharia reversa do Ghidra e implementei no VS

Code, e fiz umas ligeiras modificações para não dar problemas de execução (e reduzir linha para o print :P).

```
2  void pergunta3(void)
3
4  {
5    long in_FS_OFFSET;
6    long local_20;
7    long local_18;
8    long local_10;
9
10   local_10 = *(long *)(in_FS_OFFSET + 0x28);
11   local_18 = 0x1bd5a262d;
12   puts("3) Em qual numero eu estou pensando?");
13   __isoc99_scanf(&DAT_0040208d,&local_20);
14   if (local_18 == local_20) {
15     puts("Resposta correta!");
16   }
17   else {
18     ganhou = 0;
19   }
20   if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
21                     /* WARNING: Subroutine does not return
22     __stack_chk_fail();
23   }
24   return;
25 }
```

```
51  void pergunta3(void)
52  {
53      long local_20;
54      long local_18;
55      int ganhou;
56
57      local_18 = 0x1bd5a262d;
58      puts("3) Em qual numero eu estou pensando?");
59      __isoc99_scanf("%lu", &local_20);
70      if (local_18 == local_20) {puts("Resposta correta!");}
71      else {puts("Errou!");}
72      return;
73  }
```

Com o código acima no fundo negro, eu consegui fazer testes para verificar a resposta com a qual eu entrava. Convertendo o hexadecimal “0x1bd5a262d”, obtive 7471769133, e para verificar se essa era realmente a resposta correta, usei o código da segunda imagem acima para testar, e o resultado é o que aparece na imagem abaixo:

```
3) Em qual numero eu estou pensando?
7471769133
Resposta correta!
```