

Tutorial G-3PO

Apresentação

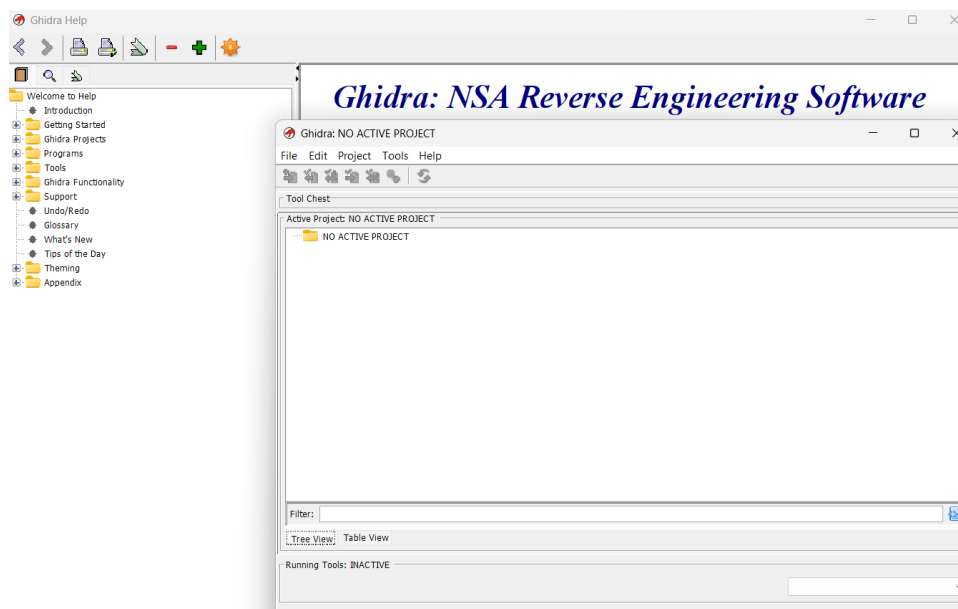
O G3PO é uma extensão para o Ghidra, um framework de engenharia reversa de software desenvolvido pela Agência de Segurança Nacional (NSA) dos Estados Unidos. Em conjunto com os modelos de linguagem da OpenAI, a extensão permite produzir comentários em linguagem natural, que explicam o funcionamento das funções encontradas no código fonte do software.

Instalação

Instale o Ghidra, que pode ser obtido neste endereço:

Faça o download do arquivo ghidra_10.4_PUBLIC_20230928.zip, ou o equivalente mais recente, e certifique-se de ter instalada a versão mais recente do Java JDK. Os arquivos de instalação podem ser obtidos aqui: <https://github.com/NationalSecurityAgency/ghidra/releases>

Efetue a instalação do Ghidra, consultando o guia de instalação, em caso de dúvidas. Concluído o processo, deve ser exibida a seguinte tela:



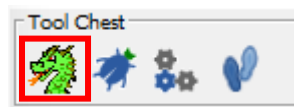
Uma vez que o G3PO utiliza os modelos de linguagem da OpenAI, será necessário obter uma chave de API. Isso pode ser feito neste link: <https://openai.com/product>

Em seguida, crie uma variável de ambiente com o nome OPENAI_API_KEY e associe sua chave de API. No Linux, isso pode ser feito por meio do comando export e, no Windows, com o comando set.

Para prosseguir, acesse os arquivos obtidos após a instalação do Ghidra e localize o ghidraRun.bat. Para iniciar a aplicação Ghidra, execute esse arquivo usando o perfil de administrador.

Concluída a instalação do Ghidra, será necessário habilitar o script G3PO. Inicie criando uma pasta Scripts dentro da pasta principal do Ghidra. Acesse o endereço [ghidra_tools/g3po at main · tenable/ghidra_tools \(github.com\)](https://github.com/tenable/ghidra_tools), obtenha uma cópia do script g3po.py e a transfira para a pasta Scripts, recém-criada.

Feito isso, no Ghidra, abra o CodeBrowser, clicando no primeiro ícone da barra Tool Chest.



Em seguida, acesse o menu Window, opção Script Manager. Na barra de ferramentas, clique no ícone Manage Script Directories.



Vamos importar o script g3po.py para o Ghidra. Faça isso clicando no ícone “Display file chooser to add bundles to list”. Localize o arquivo g3po.py e clique em OK.



A partir de agora, o script será exibido na lista de scripts e poderá ser executado na opção CodeBrowser, menu File, opção Analysis, G3PO.

Utilização

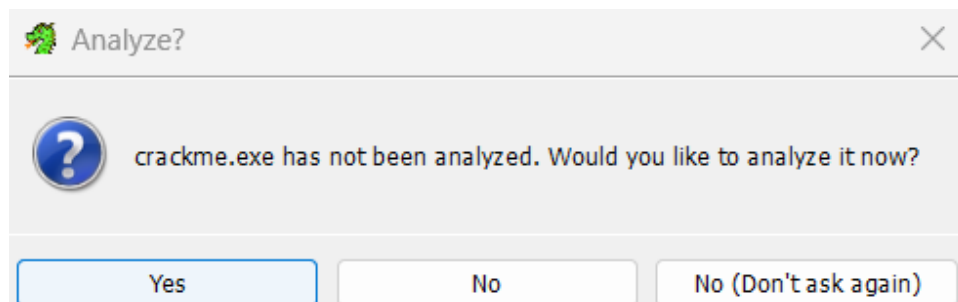
Para testar a ferramenta, precisaremos importar um arquivo executável. Disponibilizamos um arquivo de exemplo, que pode ser acessado aqui: <https://github.com/warleyandre/ia-cyber/blob/main/demos/g3po/crackme.exe>.

Trata-se de um programa desenvolvido para fins de treinamento, que apenas exibe um terminal e exige o fornecimento de uma senha.

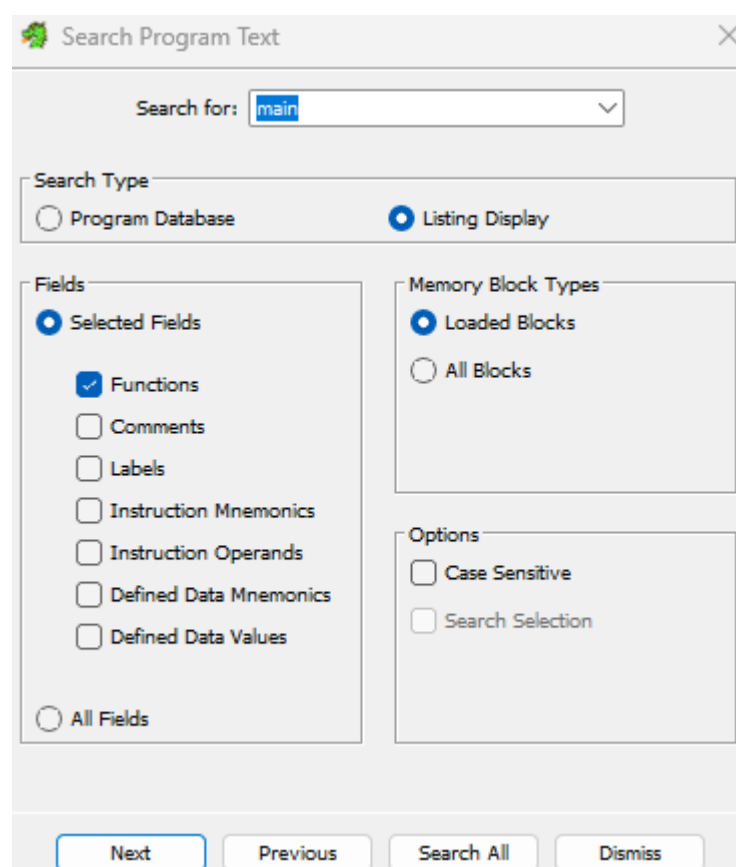
```
IOLI Crackme Level 0x00
Password:
```

Usaremos o Ghidra e o G3PO para entender o funcionamento do programa, especificamente da função que gerencia a validação da senha.

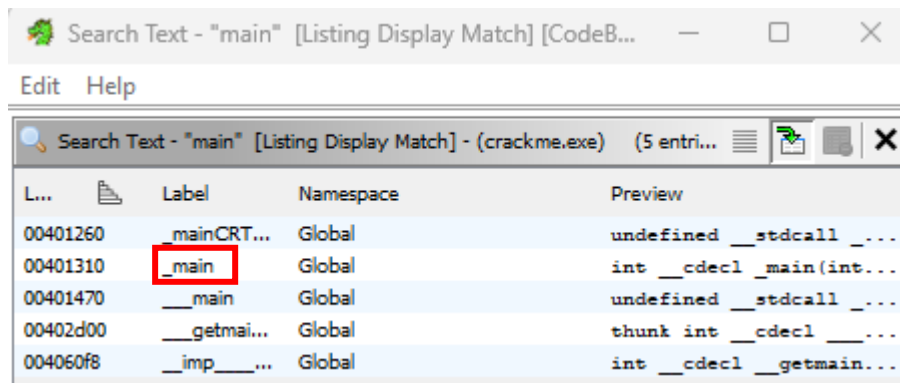
No Ghidra, abra o CodeBrowser e importe o arquivo crackme.exe, usando a opção File -> Import File. Na tela de importação, clique em OK. Em seguida, selecione Yes para iniciar a análise do arquivo executável. Confirme todas as opções seguintes.



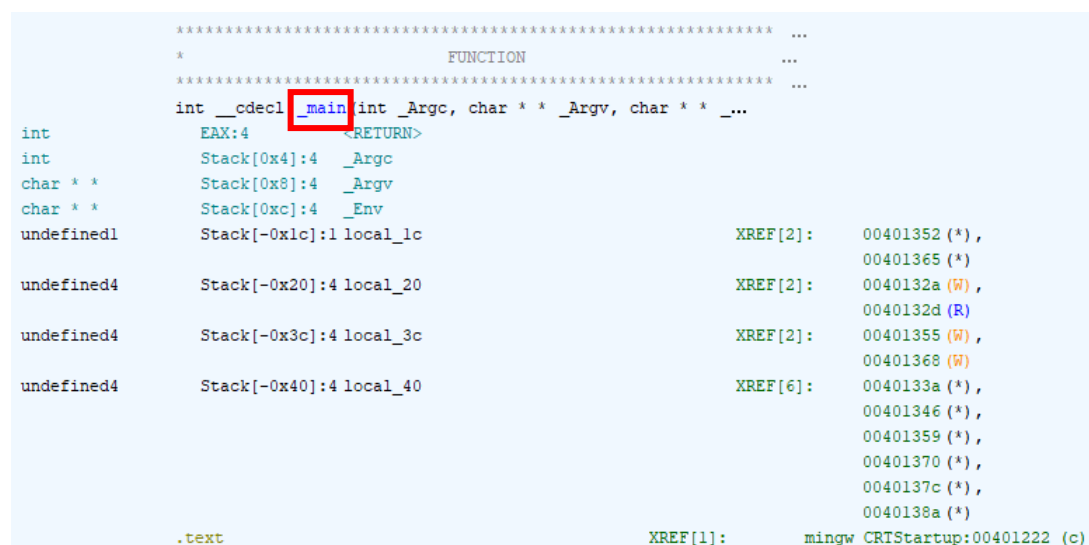
Para usar o G3PO, vamos localizar a função principal do programa. No menu Search, selecione a opção Program Text e pesquise por "main":



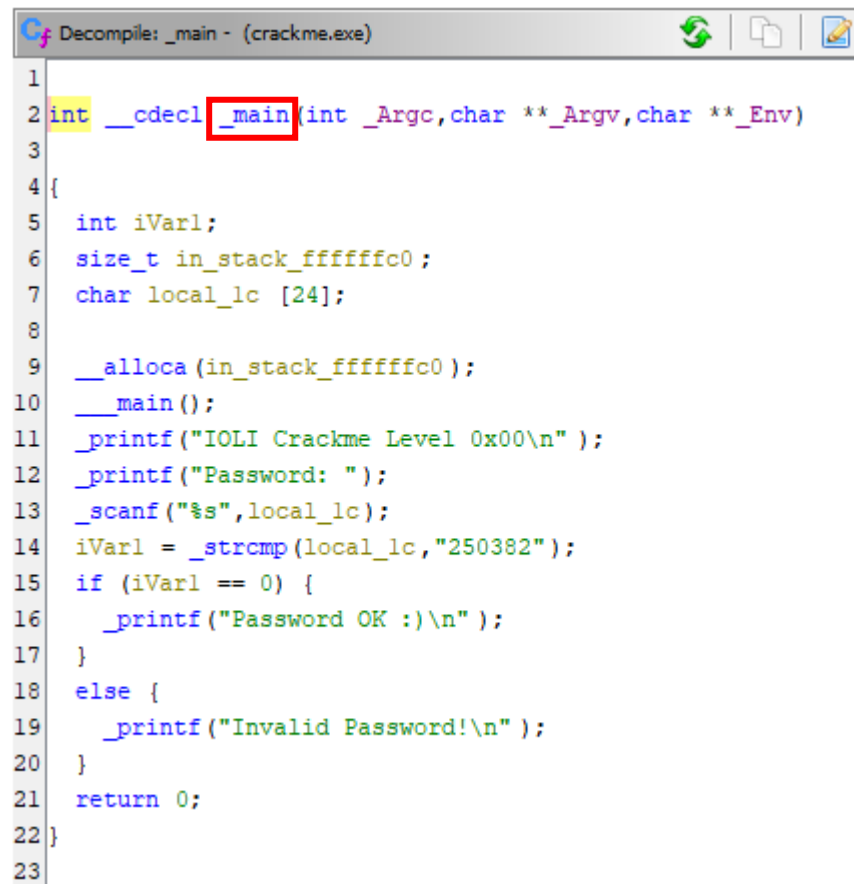
Clique em Search All e, na janela “Search Text”, selecione a opção de Label “_main” com um duplo clique do mouse:



Após o comando, a função ficará em destaque na tela principal:



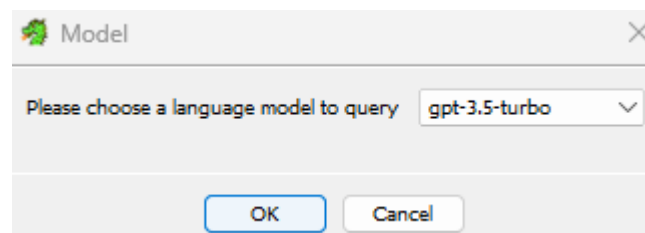
No painel “Decompile” é possível identificar o código em alto nível correspondente a essa função:



The screenshot shows a decompiler window titled "Decompile: _main - (crackme.exe)". The code is as follows:

```
1
2 int __cdecl _main(int _Argc, char **_Argv, char **_Env)
3
4 {
5     int iVar1;
6     size_t in_stack_ffffffc0;
7     char local_lc [24];
8
9     __alloca (in_stack_ffffffc0);
10    __main();
11    _printf("IOLI Crackme Level 0x00\n");
12    _printf("Password: ");
13    _scanf("%s", local_lc);
14    iVar1 = _strcmp(local_lc, "250382");
15    if (iVar1 == 0) {
16        _printf("Password OK :)\n");
17    }
18    else {
19        _printf("Invalid Password!\n");
20    }
21    return 0;
22 }
23
```

Vamos acionar o G3PO para obter uma explicação, em linguagem natural, desse código. Para isso, selecione o menu File, opção Analysis, G-3PO. Escolha o modelo que deseja utilizar e clique em OK.



Após o processamento, será gerado o texto de explicação da função, em detalhes. A seguir, o texto gerado pelo modelo gpt-3.5-turbo:

```
| 1. A função _main recebe três argumentos: _Argc, _Argv e _Env. Esses |
| argumentos são comumente usados para passar informações sobre a linha de comando |
| e o ambiente de execução para o programa. |
| 2. A variável iVarl é declarada como um inteiro. O propósito dessa variável |
| ainda não está claro. |
| 3. A variável in_stack_ffffffc0 é declarada como um tamanho_t. O propósito |
| dessa variável ainda não está claro. |
| 4. A variável local_lc é declarada como um array de caracteres com tamanho |
| 24. O propósito dessa variável ainda não está claro. |
| 5. A função __alloca é chamada com o argumento in_stack_ffffffc0. A função |
| __alloca é usada para alocar memória na pilha. O propósito dessa chamada ainda |
| não está claro. |
| 6. A função __main é chamada. Essa função é comumente chamada no início de |
| um programa em C e é responsável por inicializar o ambiente de execução. |
| 7. A função _printf é chamada para imprimir a string "IOLI Crackme Level |
| 0x00". Parece ser uma mensagem de boas-vindas ou uma descrição do programa. |
| 8. A função _printf é chamada para imprimir a string "Password: ". Parece |
| ser uma solicitação para o usuário inserir uma senha. |
| 9. A função _scanf é chamada para ler uma string digitada pelo usuário e |
| armazená-la na variável local_lc. |
| 10. A função _strcmp é chamada para comparar a string armazenada em local_lc |
| com a string "250382". O resultado da comparação é armazenado na variável iVarl. |
| 11. Se iVarl for igual a 0, a função _printf é chamada para imprimir a |
| string "Password OK :)". Caso contrário, a função _printf é chamada para |
| imprimir a string "Invalid Password!". |
| 12. A função _main retorna 0. |
| Com base na análise do código, parece que este programa é um "crackme", um |
| desafio de engenharia reversa onde o objetivo é encontrar a senha correta. O |
| programa solicita ao usuário que insira uma senha e, em seguida, compara essa |
| senha com a string "250382". Se a senha inserida for igual a "250382", o |
| programa imprime "Password OK :)", caso contrário, imprime "Invalid Password!". |
| Sugestões de nomes de variáveis: |
| - iVarl -> resultadoComparacao |
| - in_stack_ffffffc0 -> tamanhoAlocacaoPilha |
| - local_lc -> senhaInserida |
| Sugestão de nome da função: |
| _main :: crackmeLevel0x00 |
| Possíveis vulnerabilidades de segurança: |
| - O programa não realiza nenhuma validação ou verificação adicional na senha |
| inserida pelo usuário. Isso pode permitir ataques de força bruta ou tentativas |
| de adivinhar a senha. |
| - O programa não limita o tamanho da string lida pela função _scanf, o que |
| pode levar a um estouro de buffer (buffer overflow) se o usuário inserir uma |
| senha maior que 24 caracteres. Esse tipo de vulnerabilidade pode ser explorado |
| para executar código malicioso ou corromper a memória do programa. |
| |
| Model: gpt-3.5-turbo, Temperature: 0.05 |
```

Optando pelo modelo davinci-003, uma versão mais curta e simples é gerada:

```
/-----\n|AI generated comment, take with a grain of salt:\n|\n|     Esta função tem como objetivo verificar se a senha inserida pelo usuário é\n| válida. Primeiro, ela aloca memória na pilha para armazenar dados. Em seguida,\n| ela imprime uma mensagem de boas-vindas e pede que o usuário insira a senha. O\n| usuário então insere a senha e a função armazena a senha em uma variável chamada\n| "senhaInserida". A função então compara a senha inserida com a senha correta,\n| que é armazenada na variável "senhaCorreta". Se a senha inserida for igual à\n| senha correta, a função imprime uma mensagem de sucesso. Caso contrário, ela\n| imprime uma mensagem de erro.\n|\n|     $senhaInserida -> $senhaInseridaUsuario\n|\n|     $senhaCorreta -> $senhaCorretaValida\n|\n|     _main :: verificarSenha\n|\n|Model: text-davinci-003, Temperature: 0.05\n|-----\n
```

```

      /
    /~\
   |oo |
  _\=/_
 /      \
//|/.\\
||  \  ||
||  \ /  ||
#  _  _  #
   | | |
   | | |
   [| |]
   | | |
  /_  \_ \

```

** /*

Por fim, o modelo gpt4-turbo apresenta um resultado com maior riqueza de detalhes, melhor precisão e inclui análises de vulnerabilidades de segurança no código:

Parte 1:

```
| Este código é uma função principal de um programa em C que parece ser um |
| simples desafio de crackme, que é um tipo de desafio de engenharia reversa onde |
| o objetivo é encontrar uma senha ou chave para "crackear" o programa. |
| A função começa definindo algumas variáveis locais: |
| - `iVar1` é uma variável que será usada para armazenar o resultado da |
| comparação de strings. |
| - `in_stack_ffffffc0` é uma variável relacionada ao tamanho de um buffer |
| alocado dinamicamente na pilha, mas seu uso exato não é claro a partir deste |
| trecho de código. |
| - `local_lc` é um buffer de 24 caracteres que será usado para armazenar a |
| senha inserida pelo usuário. |
| O código então executa algumas ações: |
| 1. Chama `__alloca(in_stack_ffffffc0)` para alocar dinamicamente memória na |
| pilha. O tamanho dessa alocação é baseado no valor de `in_stack_ffffffc0`, que |
| não é definido neste trecho de código. Isso pode ser uma vulnerabilidade de |
| segurança se o valor de `in_stack_ffffffc0` for controlável pelo usuário ou não |
| for verificado, pois pode levar a um estouro de pilha (stack overflow). |
| 2. Chama `__main()`, que é provavelmente uma função de inicialização gerada |
| pelo compilador. |
| 3. Imprime a mensagem "IOLI Crackme Level 0x00\n" e pede ao usuário que |
| insira uma senha com a mensagem "Password: ". |
| 4. Usa `scanf` para ler uma string do usuário e armazená-la no buffer |
| `local_lc`. |
| 5. Compara a string inserida pelo usuário com a senha correta "250382" |
| usando `strcmp`. |
| 6. Se a senha estiver correta (ou seja, `strcmp` retorna 0), imprime |
| "Password OK :)\n". Caso contrário, imprime "Invalid Password!\n". |
| 7. A função retorna 0, indicando a conclusão bem-sucedida do programa. |
```


Parte 2:

```
| Sugestões de nomes de variáveis: |
|   `` |
| iVar1 -> $comparisonResult |
| local_lc -> $userInputPassword |
|   `` |
| Sugestão de nome para a função: |
|   `` |
| _main :: $checkPassword |
|   `` |
| Vulnerabilidades de segurança: |
| 1. O uso de `scanf` com o especificador de formato `%s` para ler a senha é |
| inseguro, pois pode levar a um estouro de buffer se o usuário inserir uma string |
| maior que 23 caracteres (o 24º é para o caractere nulo de terminação). Isso |
| poderia ser explorado para sobrescrever outras partes da pilha e potencialmente |
| assumir o controle do fluxo do programa. |
| 2. A alocação dinâmica de memória na pilha com `__alloca` usando uma |
| variável não inicializada (`in_stack_ffffffc0`) é potencialmente perigosa. Se o |
| valor dessa variável for grande o suficiente, pode causar um estouro de pilha. |
| Para corrigir a vulnerabilidade de segurança relacionada ao `scanf`, deve-se |
| usar uma versão mais segura como `fgets` ou limitar o número de caracteres lidos |
| com `scanf("%23s", $userInputPassword);`. Quanto à alocação dinâmica com |
| `__alloca`, seria necessário entender melhor o contexto do código para fornecer |
| uma solução adequada. |
| |
| Model: gpt-4-1106-preview, Temperature: 0.05 |
|-----/
```

```
/
/~\
|oo |
_\/_
/ \
//|/.\\
|| \_||
|| \| ||
# \_/_ #
| | |
| | |
[]|[]
| | |
/_|_|
```