Shriya Pingulkar – 16010420078
Diti Divekar - 16010420091

IA – 1 Report

Ghidra

Ghidra is a powerful and free reverse engineering tool developed by the US National Security Agency (NSA). Reverse engineering is the process of dissecting software (in the form of executable files) to understand its underlying structure and functionality. Ghidra translates the machine code (binary) of software back into a human-readable format, allowing analysts to review, analyze, and understand the software's operations.

Installation:

Windows:

1. Prerequisite: Before installing Ghidra, ensure you have JDK 11 installed. If not, download and install it from the official Oracle website.
2. Visit the official Ghidra website and download the latest version.
3. Once downloaded, unpack the file.
4. Double-click on “runGhidra.bat” to launch Ghidra.
5. Upon the initial startup, you'll be presented with a user agreement. Proceed by accepting it.

MacOS:

1. Download and extract the launcher AppleScript template app below. Optionally modify Ghidra.app/Info.plist to your liking.
2. Download the latest OpenJDK and extract it to Ghidra.app/jdk. Make sure Ghidra.app/jdk/Contents/Home/bin/java exists.
3. Download the latest Ghidra and extract it to Ghidra.app/ghidra. Make sure Ghidra.app/ghidra/ghidraRun exists.
4. Copy Ghidra.app to your Applications directory.

Or

1. Use the prebuild to install Ghidra.

Objective

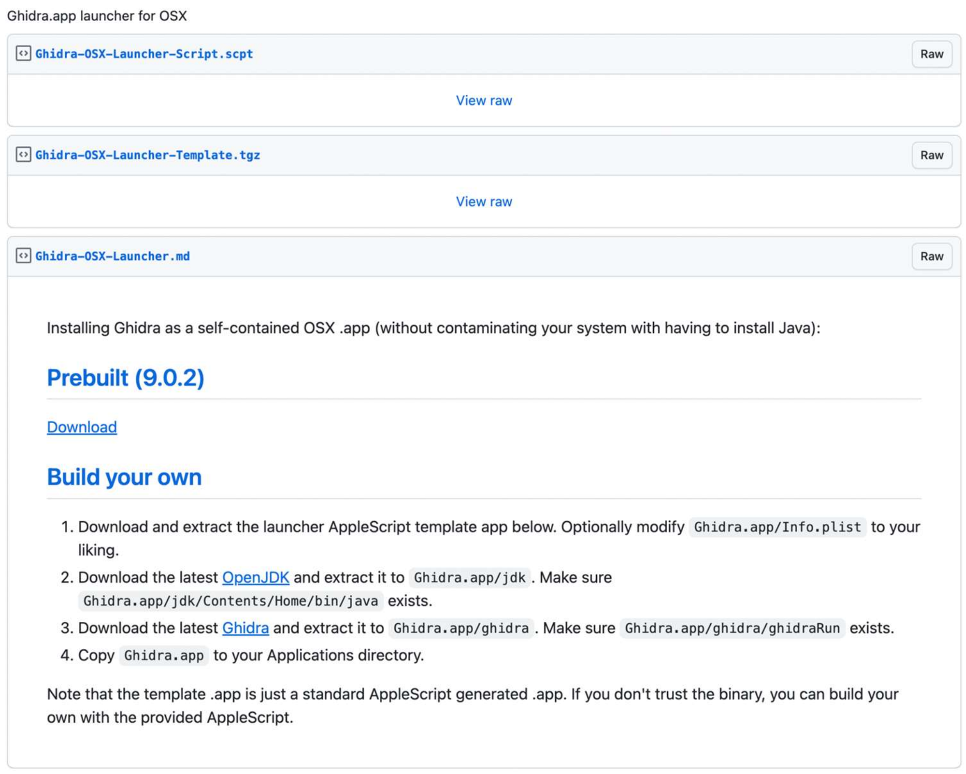
The goal of this report is to provide a clear and systematic tutorial on using Ghidra to analyze and reverse engineer the CrackMe0x00 executable, facilitating a deeper understanding of software behavior and security vulnerabilities.
CrackMe0x00: CrackMe0x00 is a type of challenge used in the cybersecurity field to test and hone reverse engineering skills. It's essentially a program that prompts users for a password. The objective is to determine the correct password without being explicitly told, typically by examining the software's code.

Mock Scenario

A corporate cybersecurity team detects a suspicious binary on their network. Preliminary analysis suggests that this binary may be associated with an Advanced Persistent Threat (APT) campaign. The binary's behavior and origin must be confirmed. The team chooses crackme0x00 as a representative sample for binary analysis due to its known behavior and structure, ensuring they can effectively navigate and interpret results in Ghidra.

Step-by-Step Guide

Step 1: Installing Ghidra from github


Ghidra.app launcher for OSX

Ghidra-OSX-Launcher-Script.scpt

Ghidra-OSX-Launcher-Template.tgz

Ghidra-OSX-Launcher.md

Installing Ghidra as a self-contained OSX .app (without contaminating your system with having to install Java):

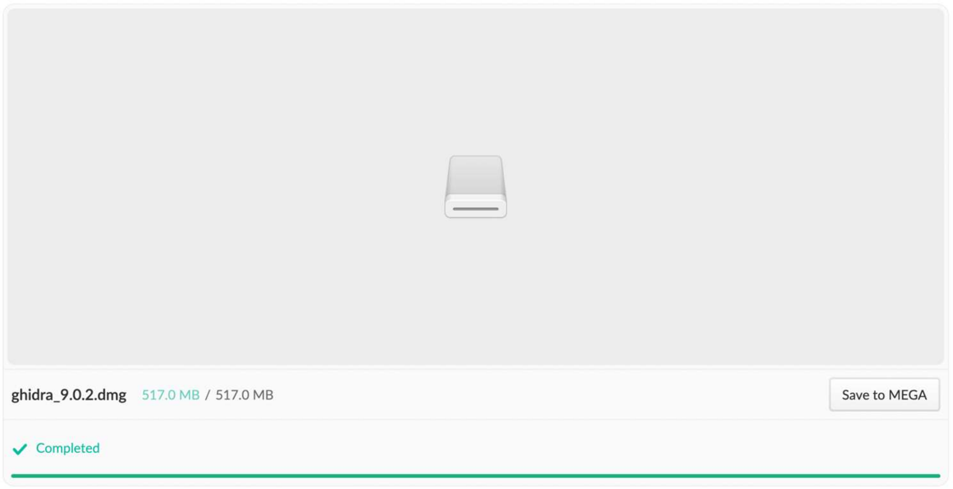
Prebuilt (9.0.2)

Download

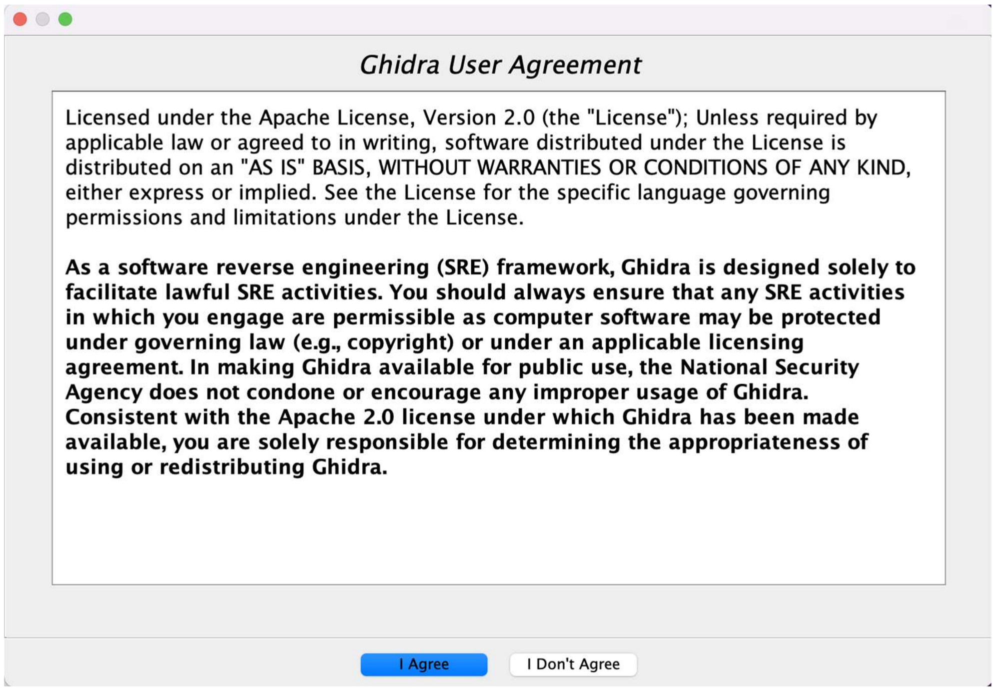
Build your own

1. Download and extract the launcher AppleScript template app below. Optionally modify `Ghidra.app/Info.plist` to your liking.
2. Download the latest OpenJDK and extract it to `Ghidra.app/jdk`. Make sure `Ghidra.app/jdk/Contents/Home/bin/java` exists.
3. Download the latest Ghidra and extract it to `Ghidra.app/ghidra`. Make sure `Ghidra.app/ghidra/ghidraRun` exists.
4. Copy `Ghidra.app` to your Applications directory.

Note that the template .app is just a standard AppleScript generated .app. If you don't trust the binary, you can build your own with the provided AppleScript.



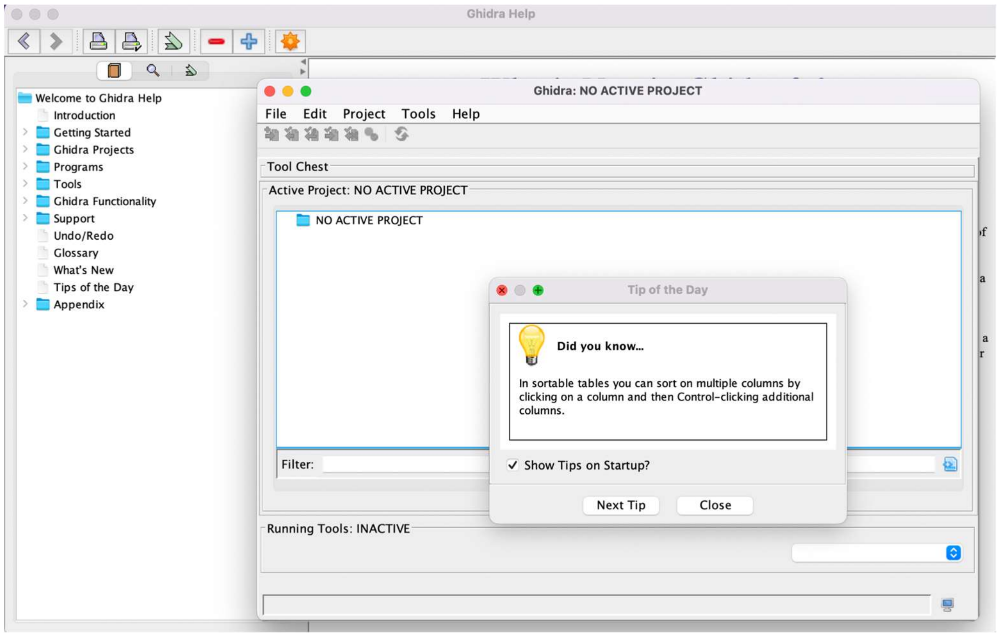
Step 2: Launching Ghidra



Ghidra User Agreement

Licensed under the Apache License, Version 2.0 (the "License"); Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

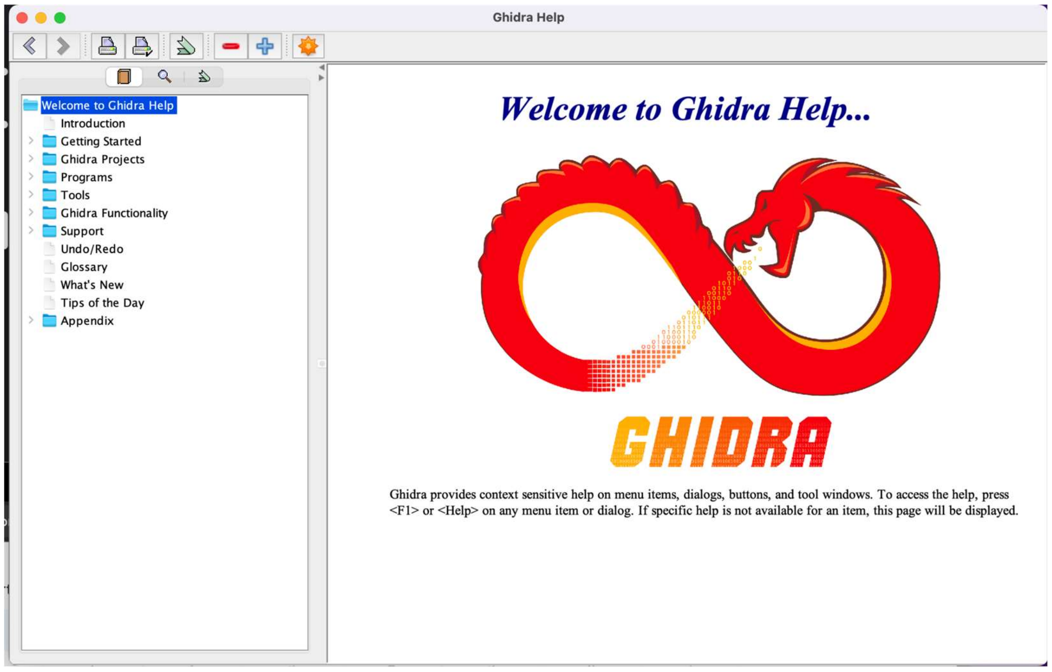
As a software reverse engineering (SRE) framework, Ghidra is designed solely to facilitate lawful SRE activities. You should always ensure that any SRE activities in which you engage are permissible as computer software may be protected under governing law (e.g., copyright) or under an applicable licensing agreement. In making Ghidra available for public use, the National Security Agency does not condone or encourage any improper usage of Ghidra. Consistent with the Apache 2.0 license under which Ghidra has been made available, you are solely responsible for determining the appropriateness of using or redistributing Ghidra.



Did you know...

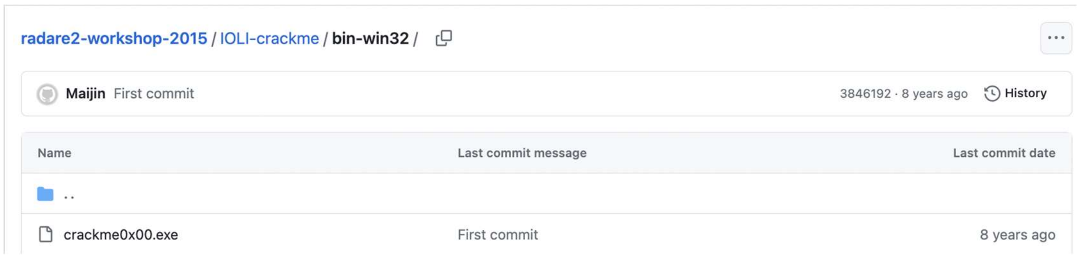
In sortable tables you can sort on multiple columns by clicking on a column and then Control-clicking additional columns.

✓ Show Tips on Startup?



Ghidra provides context sensitive help on menu items, dialogs, buttons, and tool windows. To access the help, press <F1> or <Help> on any menu item or dialog. If specific help is not available for an item, this page will be displayed.

Step 3: Download the CrackMe0x00.exe file from the provided repository (https://github.com/Maijin/Workshop2015/tree/master/IOLI-crackme/bin-win32).

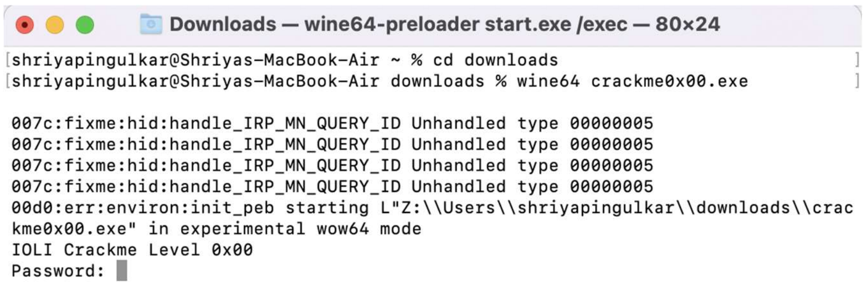


radare2-workshop-2015 / IOLI-crackme / bin-win32 /

Maijin First commit 3846192 · 8 years ago History

Name	Last commit message	Last commit date
..		
crackme0x00.exe	First commit	8 years ago

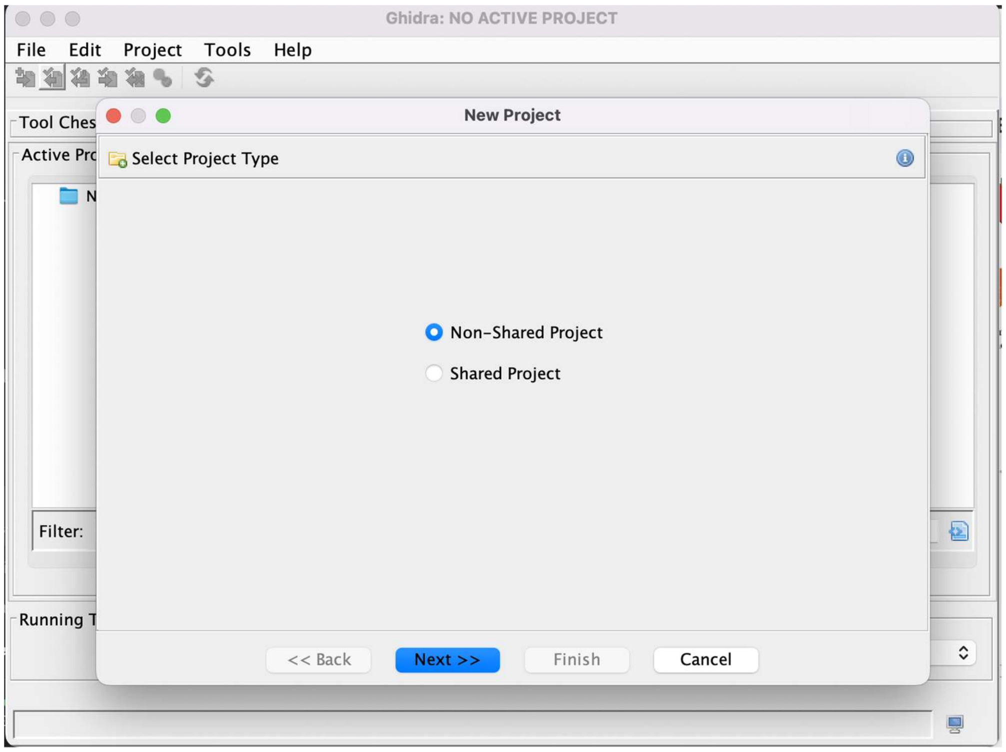
Step 4: Attempt to open and run the file.



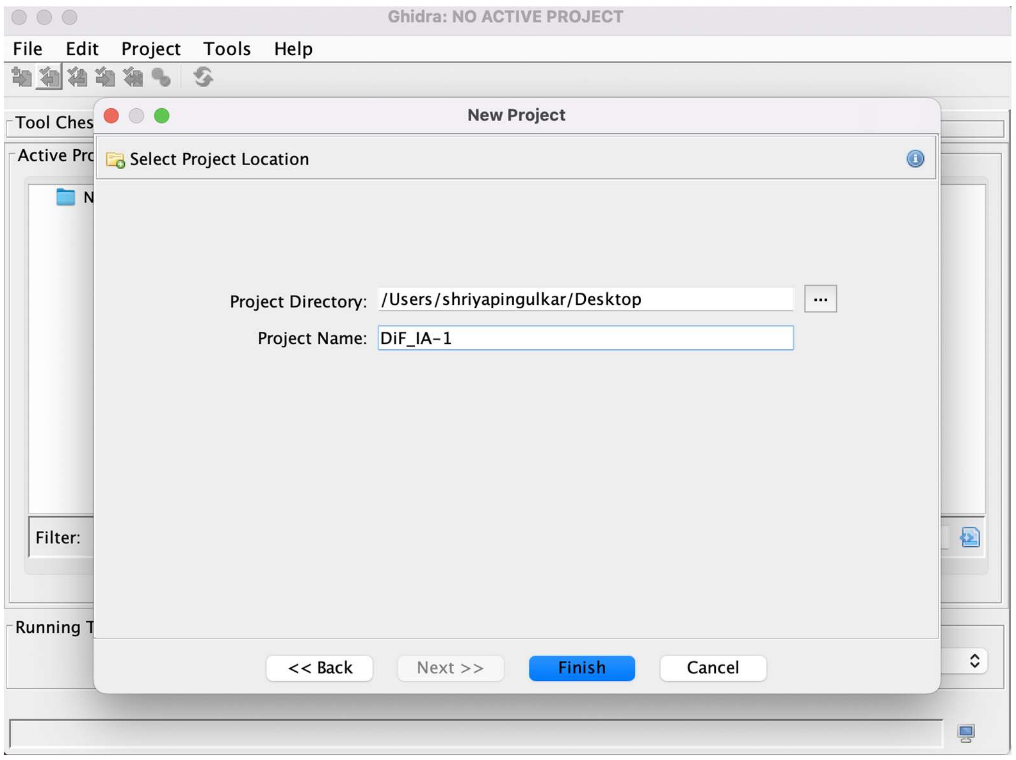
Downloads — wine64-preloader start.exe /exec — 80×24

```
shriyapingulkar@Shriyas-MacBook-Air ~ % cd downloads
shriyapingulkar@Shriyas-MacBook-Air downloads % wine64 crackme0x00.exe

007c:fixme:hid:handle_IRP_MN_QUERY_ID Unhandled type 00000005
007c:fixme:hid:handle_IRP_MN_QUERY_ID Unhandled type 00000005
007c:fixme:hid:handle_IRP_MN_QUERY_ID Unhandled type 00000005
007c:fixme:hid:handle_IRP_MN_QUERY_ID Unhandled type 00000005
00d0:err:environ:init_peb starting L"Z:\\Users\\shriyapingulkar\\downloads\\crac
kme0x00.exe" in experimental wow64 mode
IOLI Crackme Level 0x00
Password:
```

Step 5: initiate a new project

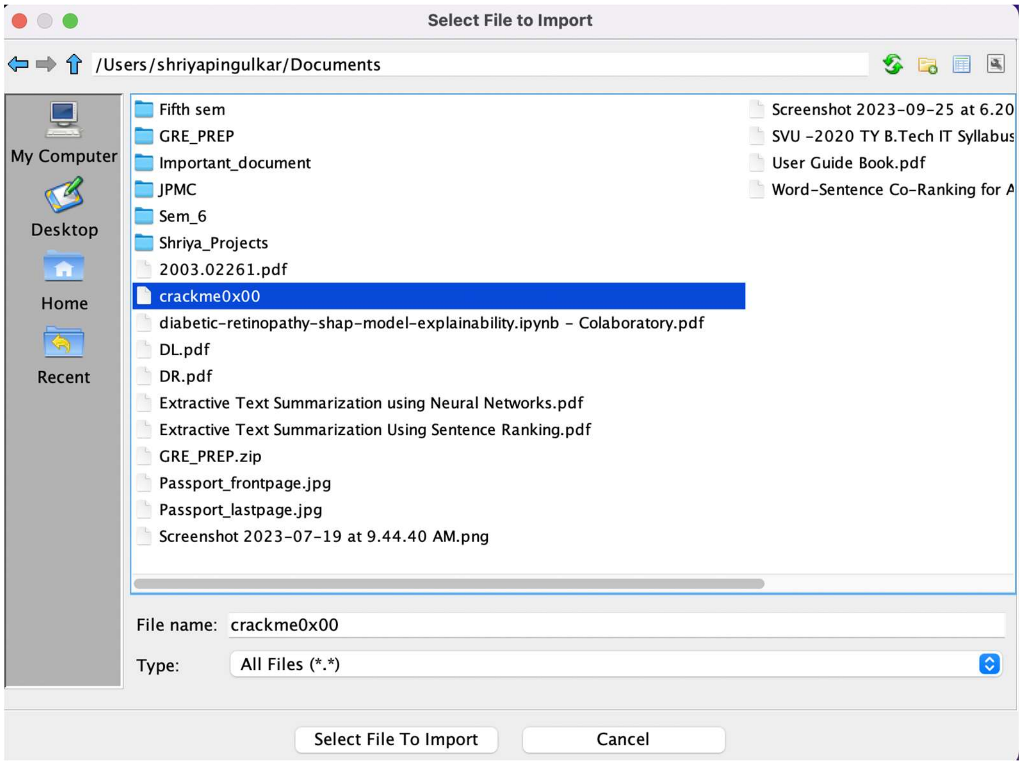


Non-Shared Project [✓]
Shared Project



Project Directory: /Users/shriyapingulkar/Desktop
Project Name: DiF_IA-1

Step 6: Import the CrackMe0x00.exe file into your Ghidra project. This can be achieved by either dragging & dropping the file into Ghidra or using the "File -> Import File" option.

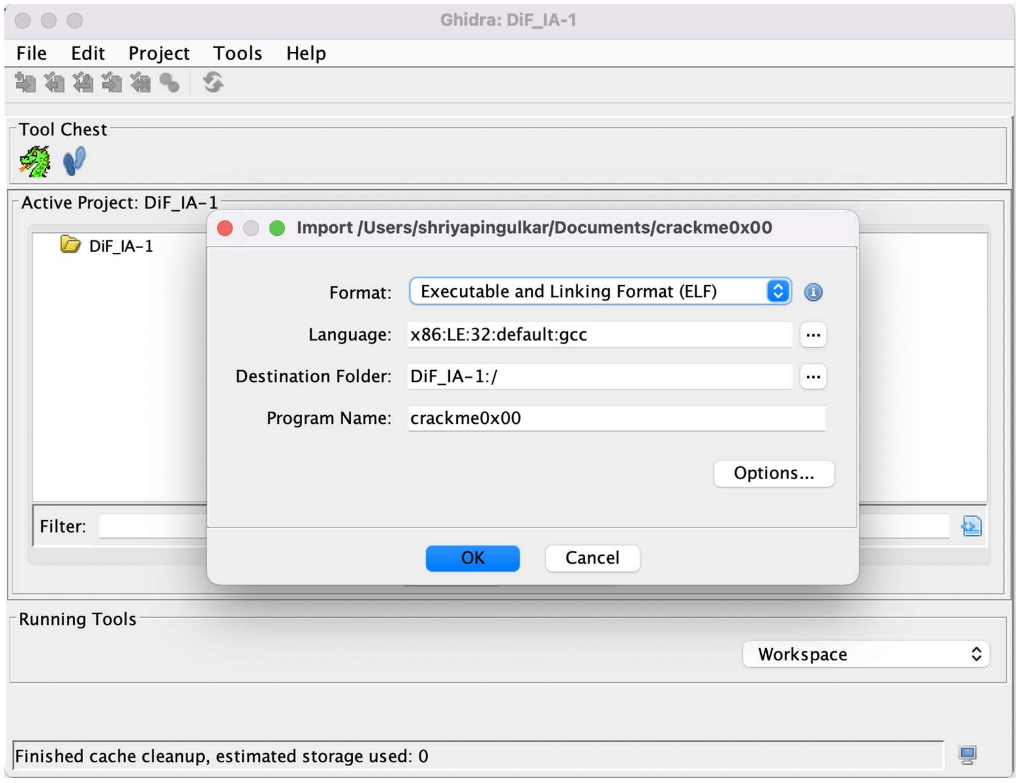

Select File to Import

/Users/shriyapingulkar/Documents

- Fifth sem
- GRE_PREP
- Important_document
- JPMC
- Sem_6
- Shriya_Projects
- 2003.02261.pdf
- crackme0x00
- diabetic-retinopathy-shap-model-explainability.ipynb - Colaboratory.pdf
- DL.pdf
- DR.pdf
- Extractive Text Summarization using Neural Networks.pdf
- Extractive Text Summarization Using Sentence Ranking.pdf
- GRE_PREP.zip
- Passport_frontpage.jpg
- Passport_lastpage.jpg
- Screenshot 2023-07-19 at 9.44.40 AM.png
- Screenshot 2023-09-25 at 6.20
- SVU -2020 TY B.Tech IT Syllabus
- User Guide Book.pdf
- Word-Sentence Co-Ranking for A

File name: crackme0x00

Type: All Files (*.*)


Ghidra: DiF_IA-1

Import /Users/shriyapingulkar/Documents/crackme0x00

Format: Executable and Linking Format (ELF)

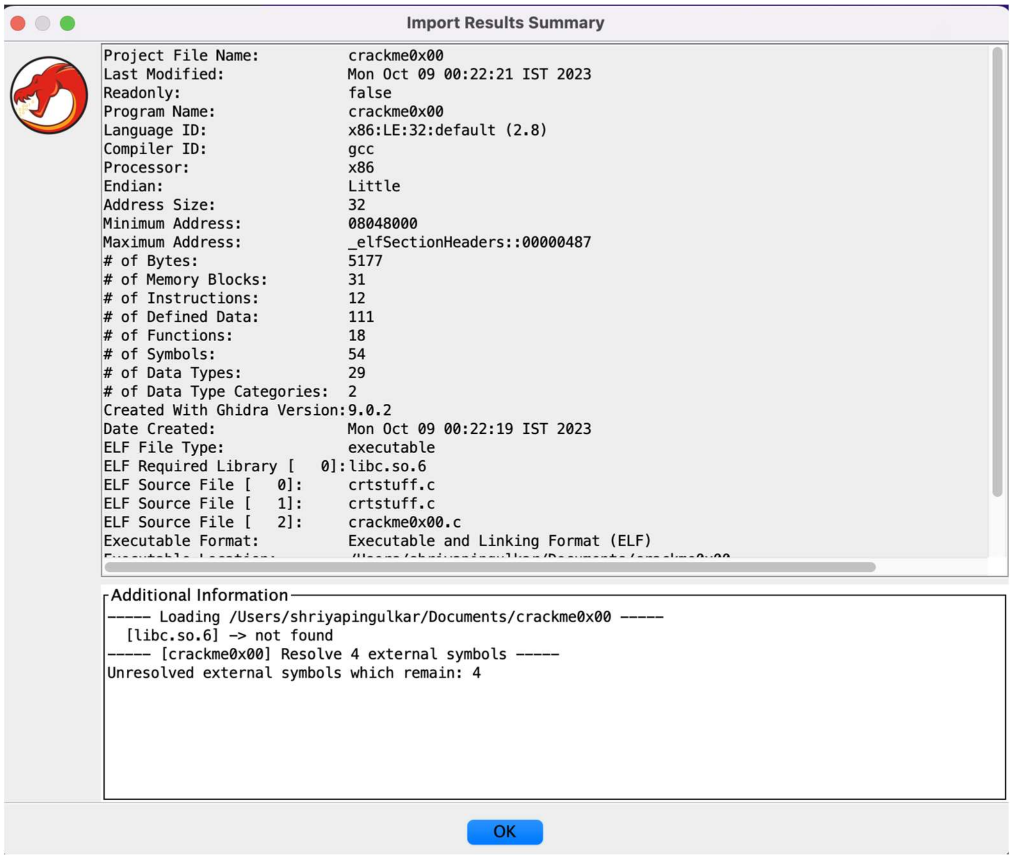
Language: x86:LE:32:default:gcc

Destination Folder: DiF_IA-1:/

Program Name: crackme0x00

Finished cache cleanup, estimated storage used: 0

Step 7: Import result summary is displayed

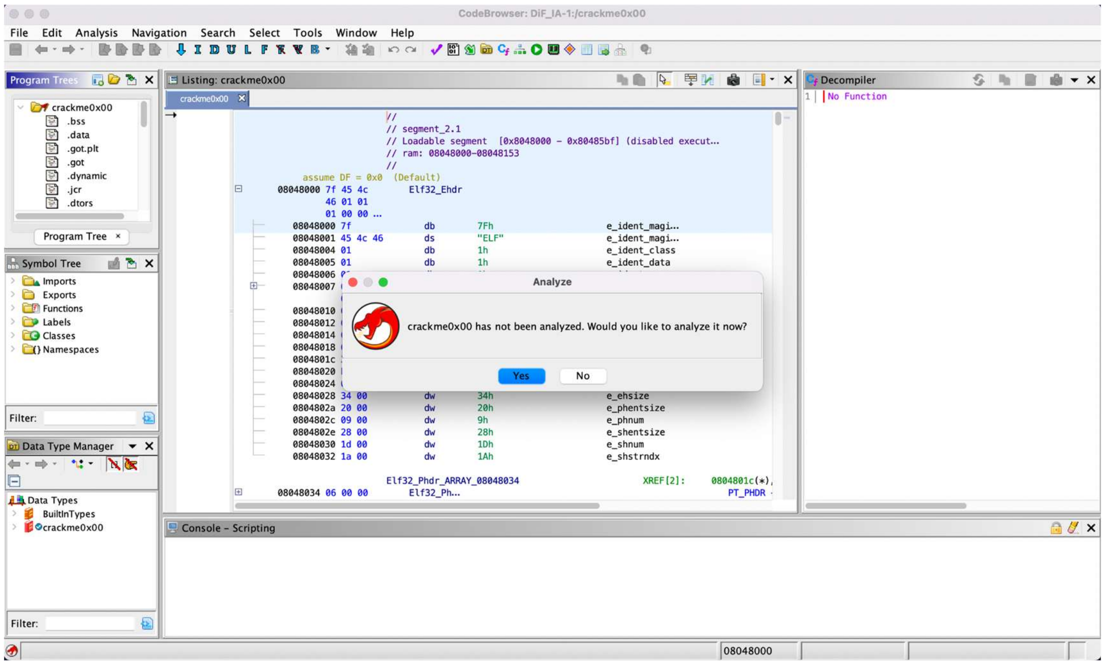


```
Project File Name:           crackme0x00
Last Modified:               Mon Oct 09 00:22:21 IST 2023
Readonly:                    false
Program Name:                crackme0x00
Language ID:                 x86:LE:32:default (2.8)
Compiler ID:                 gcc
Processor:                   x86
Endian:                      Little
Address Size:                32
Minimum Address:             08048000
Maximum Address:             _elfSectionHeaders::00000487
# of Bytes:                  5177
# of Memory Blocks:          31
# of Instructions:           12
# of Defined Data:           111
# of Functions:              18
# of Symbols:                54
# of Data Types:             29
# of Data Type Categories:   2
Created With Ghidra Version:9.0.2
Date Created:                Mon Oct 09 00:22:19 IST 2023
ELF File Type:               executable
ELF Required Library [   0]:libc.so.6
ELF Source File [   0]:      crtstuff.c
ELF Source File [   1]:      crtstuff.c
ELF Source File [   2]:      crackme0x00.c
Executable Format:           Executable and Linking Format (ELF)
```

Additional Information

```
----- Loading /Users/shriyapingulkar/Documents/crackme0x00 -----
  [libc.so.6] -> not found
----- [crackme0x00] Resolve 4 external symbols -----
Unresolved external symbols which remain: 4
```

Step 8: Upon prompt select “Yes” to analyze the binary file.

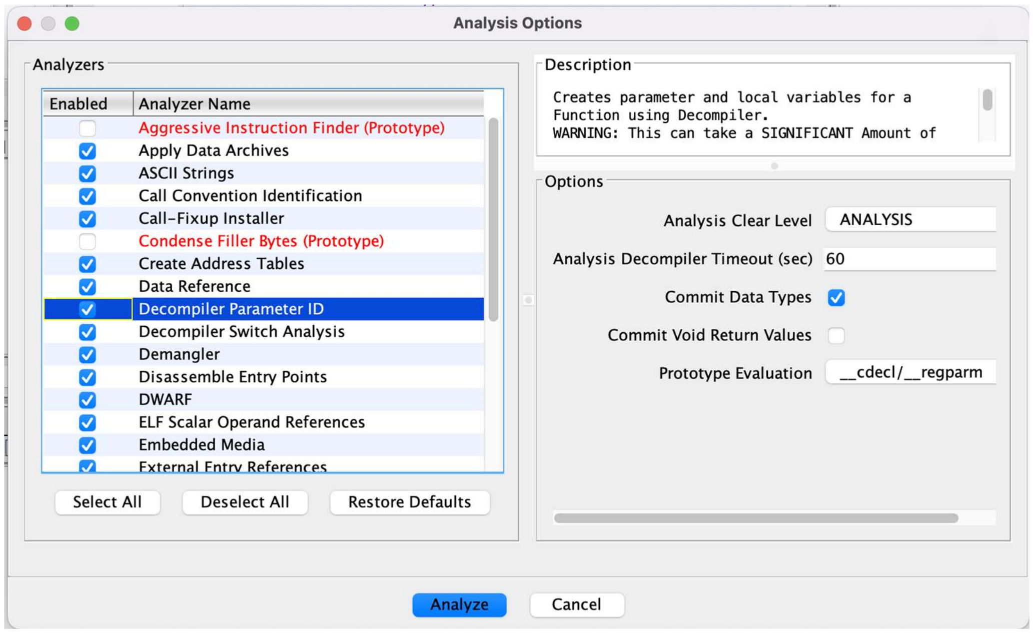


```
                               //
                               // segment_2.1
                               // Loadable segment  [0x8048000 - 0x80485bf] (disabled execut...
                               // ram: 08048000-08048153
                               //
             assume DF = 0x0  (Default)
     08048000 7f 45 4c        Elf32_Ehdr
              46 01 01
              01 00 00 ...
        08048000 7f             db        7Fh                e_ident_magi...
        08048001 45 4c 46       ds        "ELF"              e_ident_magi...
        08048004 01             db        1h                 e_ident_class
        08048005 01             db        1h                 e_ident_data
        08048006
        08048007
        08048010
        08048012
        08048014
        08048018
        0804801c
        08048020
        08048024
        08048028 34 00          dw        34h                e_ehsize
        0804802a 20 00          dw        20h                e_phentsize
        0804802c 09 00          dw        9h                 e_phnum
        0804802e 28 00          dw        28h                e_shentsize
        08048030 1d 00          dw        1Dh                e_shnum
        08048032 1a 00          dw        1Ah                e_shstrndx

                               Elf32_Phdr_ARRAY_08048034          XREF[2]:     0804801c(*),
     08048034 06 00 00        Elf32_Ph...                                      PT_PHDR
```

Analyze

crackme0x00 has not been analyzed. Would you like to analyze it now?

Step 9: Ghidra will then display different analysis types available. Default settings are recommended for this project with the addition of ‘Decompiler Parameter ID’



Analysis Options

Analyzers

Enabled	Analyzer Name
	Aggressive Instruction Finder (Prototype)
✓	Apply Data Archives
✓	ASCII Strings
✓	Call Convention Identification
✓	Call-Fixup Installer
	Condense Filler Bytes (Prototype)
✓	Create Address Tables
✓	Data Reference
✓	Decompiler Parameter ID
✓	Decompiler Switch Analysis
✓	Demangler
✓	Disassemble Entry Points
✓	DWARF
✓	ELF Scalar Operand References
✓	Embedded Media
✓	External Entry References

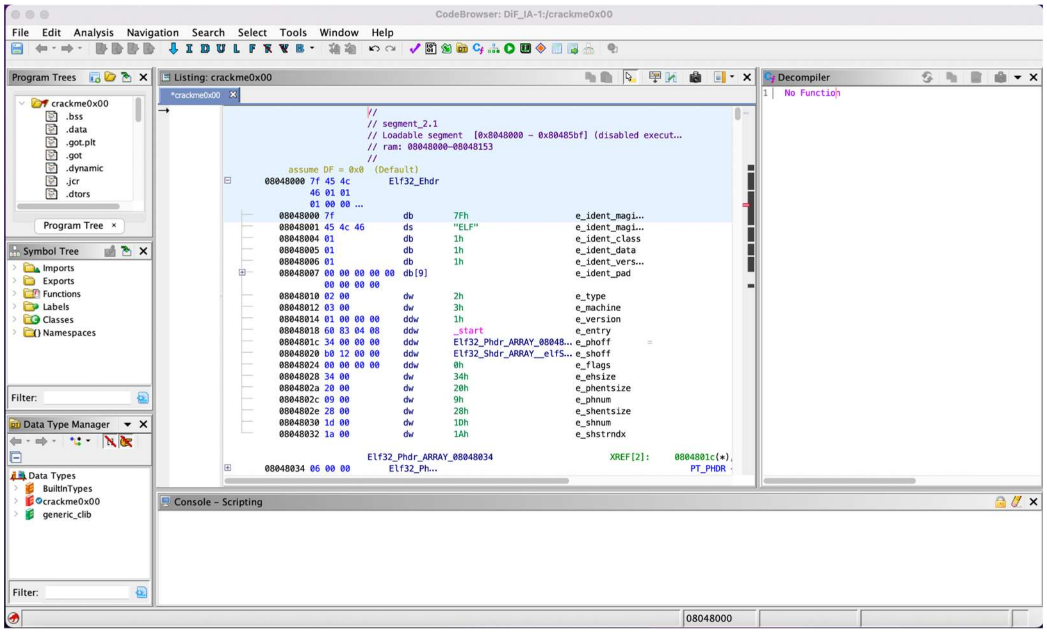
Description

```
Creates parameter and local variables for a
Function using Decompiler.
WARNING: This can take a SIGNIFICANT Amount of
```

Options

Analysis Clear Level	ANALYSIS
Analysis Decompiler Timeout (sec)	60
Commit Data Types	✓
Commit Void Return Values	
Prototype Evaluation	__cdecl/__regparm

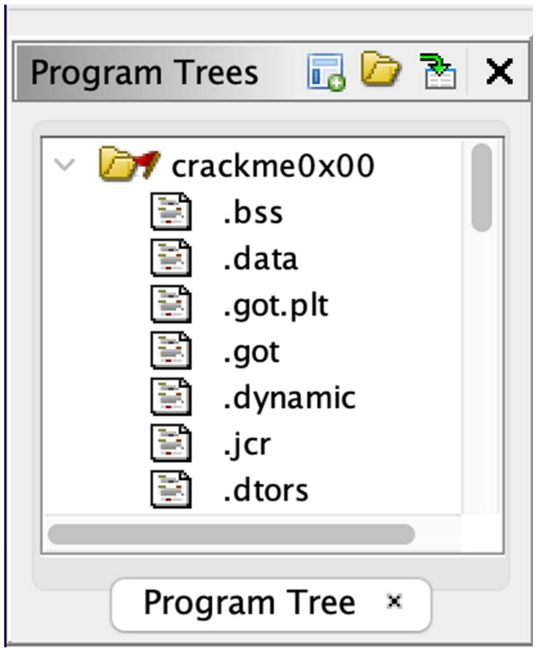
Step 10: After Ghidra completes the analysis, the main code browser window will emerge.



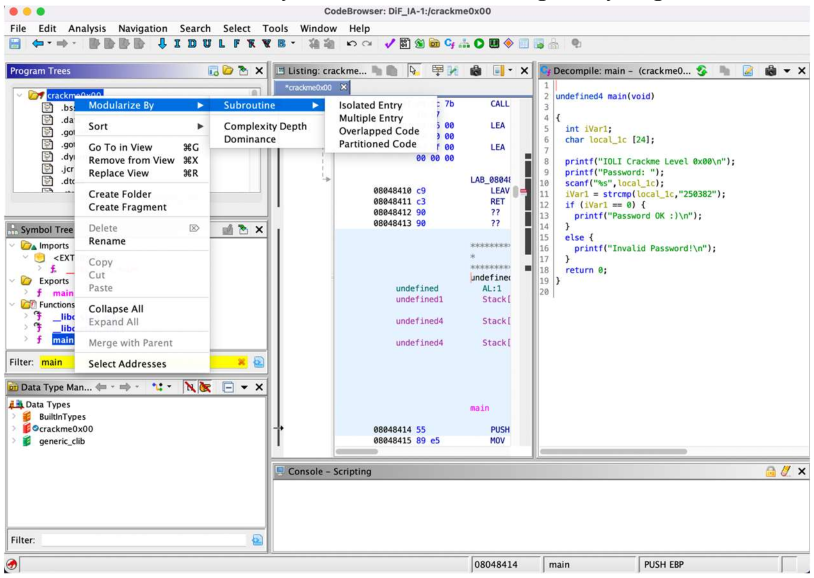
```
                    //
                    // segment_2.1
                    // Loadable segment  [0x8048000 - 0x80485bf] (disabled execut…
                    // ram: 08048000-08048153
                    //
       assume DF = 0x0  (Default)
    08048000 7f 45 4c        Elf32_Ehdr
             46 01 01
             01 00 00 …
       08048000 7f          db       7Fh                  e_ident_magi…
       08048001 45 4c 46    ds       "ELF"                e_ident_magi…
       08048004 01          db       1h                   e_ident_class
       08048005 01          db       1h                   e_ident_data
       08048006 01          db       1h                   e_ident_vers…
       08048007 00 00 00 00 00 db[9]                      e_ident_pad
                00 00 00 00
       08048010 02 00       dw       2h                   e_type
       08048012 03 00       dw       3h                   e_machine
       08048014 01 00 00 00 ddw      1h                   e_version
       08048018 60 83 04 08 ddw      _start               e_entry
       0804801c 34 00 00 00 ddw      Elf32_Phdr_ARRAY_08048… e_phoff
       08048020 b0 12 00 00 ddw      Elf32_Shdr_ARRAY__elfS… e_shoff
       08048024 00 00 00 00 ddw      0h                   e_flags
       08048028 34 00       dw       34h                  e_ehsize
       0804802a 20 00       dw       20h                  e_phentsize
       0804802c 09 00       dw       9h                   e_phnum
       0804802e 28 00       dw       28h                  e_shentsize
       08048030 1d 00       dw       1Dh                  e_shnum
       08048032 1a 00       dw       1Ah                  e_shstrndx

                    Elf32_Phdr_ARRAY_08048034                 XREF[2]:     0804801c(*),
    08048034 06 00 00        Elf32_Ph…                                    PT_PHDR
```

Step 11: Exploring Ghidra’s Main Windows:

1. Program Trees: Helps manage reverse engineering code sections.



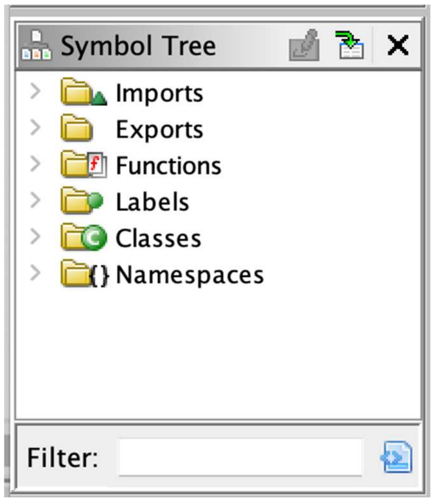
   a. Users can modularize by "Subroutine", "Complexity depth", or "Dominance".



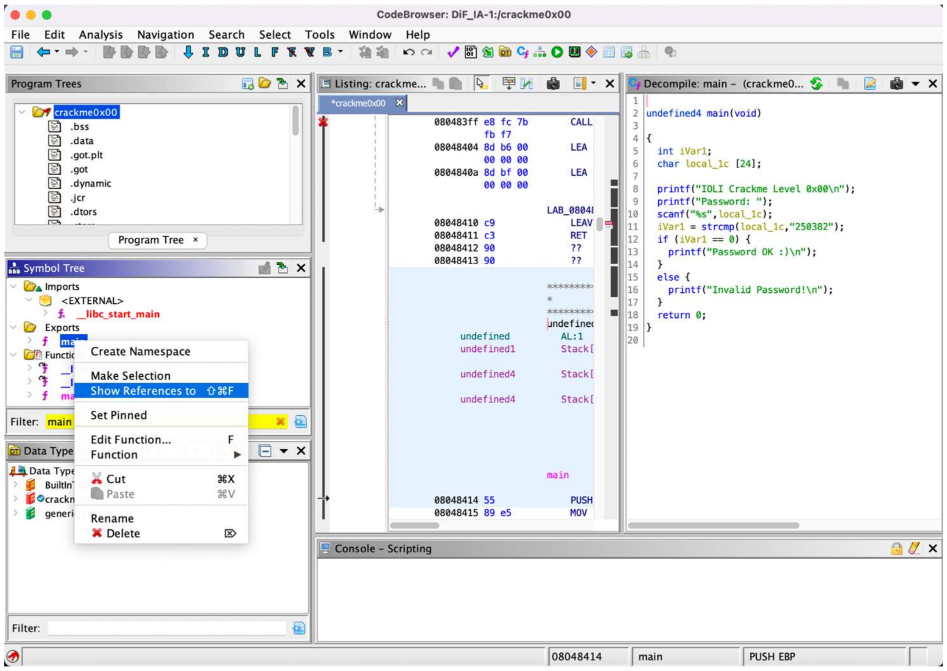
```
08048410 c9        LEAV
08048411 c3        RET
08048412 90        ??
08048413 90        ??
08048414 55        PUSH
08048415 89 e5     MOV
```

```
undefined4 main(void)

{
  int iVar1;
  char local_1c [24];

  printf("IOLI Crackme Level 0x00\n");
  printf("Password: ");
  scanf("%s",local_1c);
  iVar1 = strcmp(local_1c,"250382");
  if (iVar1 == 0) {
    printf("Password OK :)\n");
  }
  else {
    printf("Invalid Password!\n");
  }
  return 0;
}
```

2. Symbol Tree: Useful for viewing different aspects of a binary file, including import, export, functions, labels, classes, and namespaces.



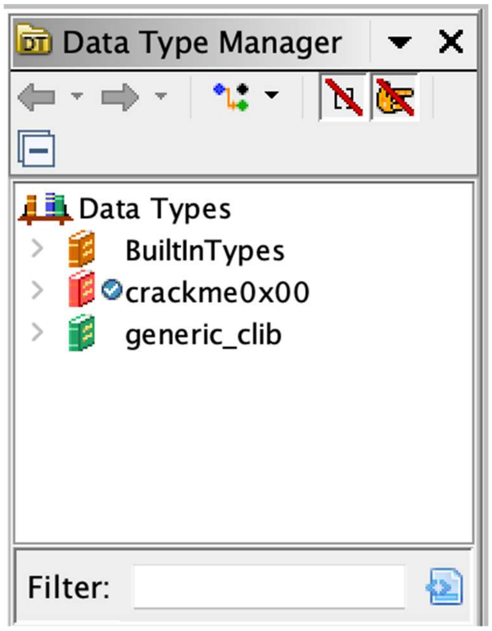
a. The "Import" section reveals various DLL libraries and their respective references.



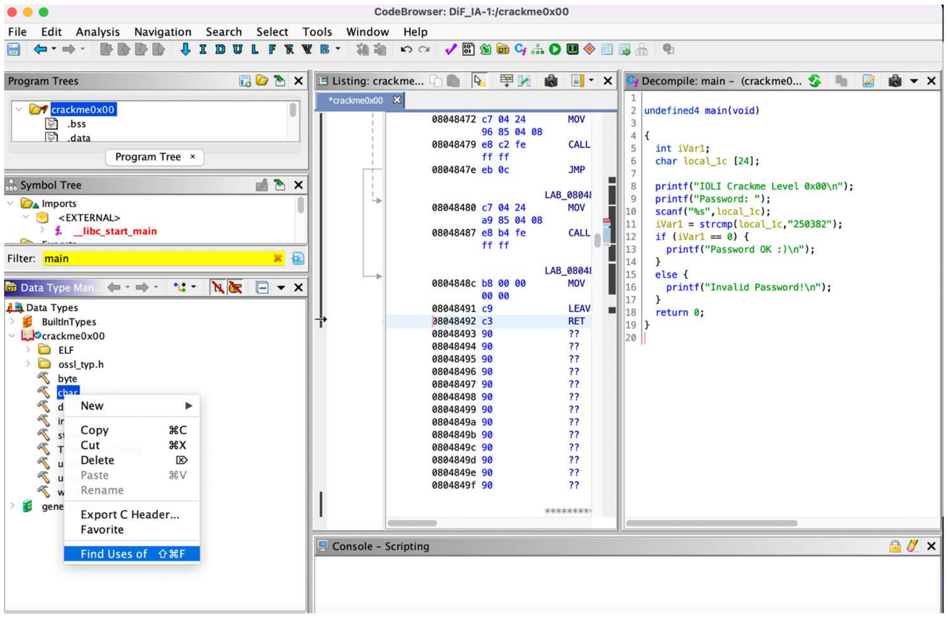
```
080483ff e8 fc 7b       CALL
         fb f7
08048404 8d b6 00       LEA
         00 00 00
0804840a 8d bf 00       LEA
         00 00 00

                     LAB_0804
08048410 c9             LEAV
08048411 c3             RET
08048412 90             ??
08048413 90             ??

                     *********
                     *
                     *********
                     undefined
      undefined         AL:1
      undefined1        Stack[

      undefined4        Stack[

      undefined4        Stack[



                     main

08048414 55             PUSH
08048415 89 e5          MOV
```

```c
undefined4 main(void)

{
  int iVar1;
  char local_1c [24];

  printf("IOLI Crackme Level 0x00\n");
  printf("Password: ");
  scanf("%s",local_1c);
  iVar1 = strcmp(local_1c,"250382");
  if (iVar1 == 0) {
    printf("Password OK :)\n");
  }
  else {
    printf("Invalid Password!\n");
  }
  return 0;
}
```

3. Data Type Manager:

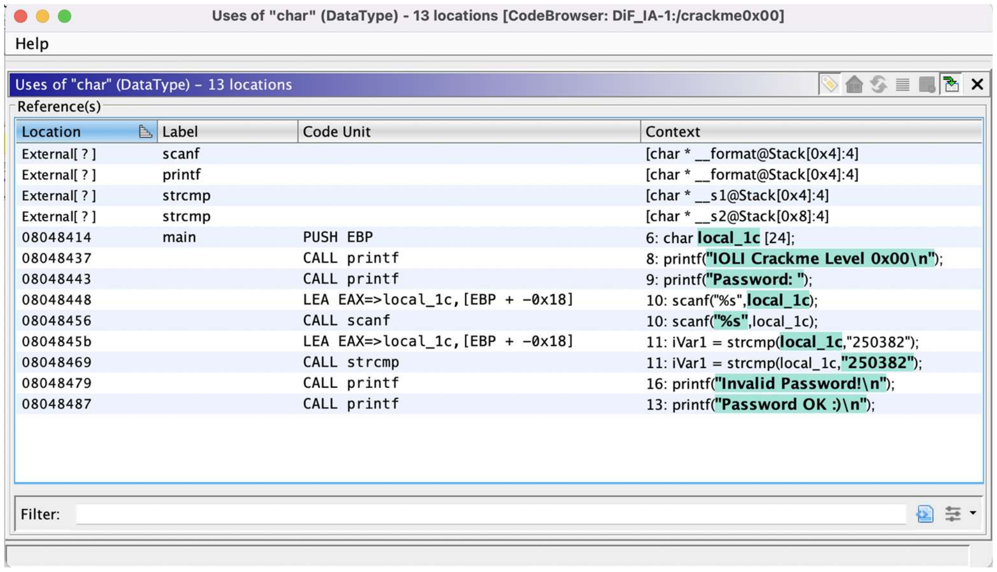


a. Navigate to the "Defined Data Types" window to inspect and manage different data types.



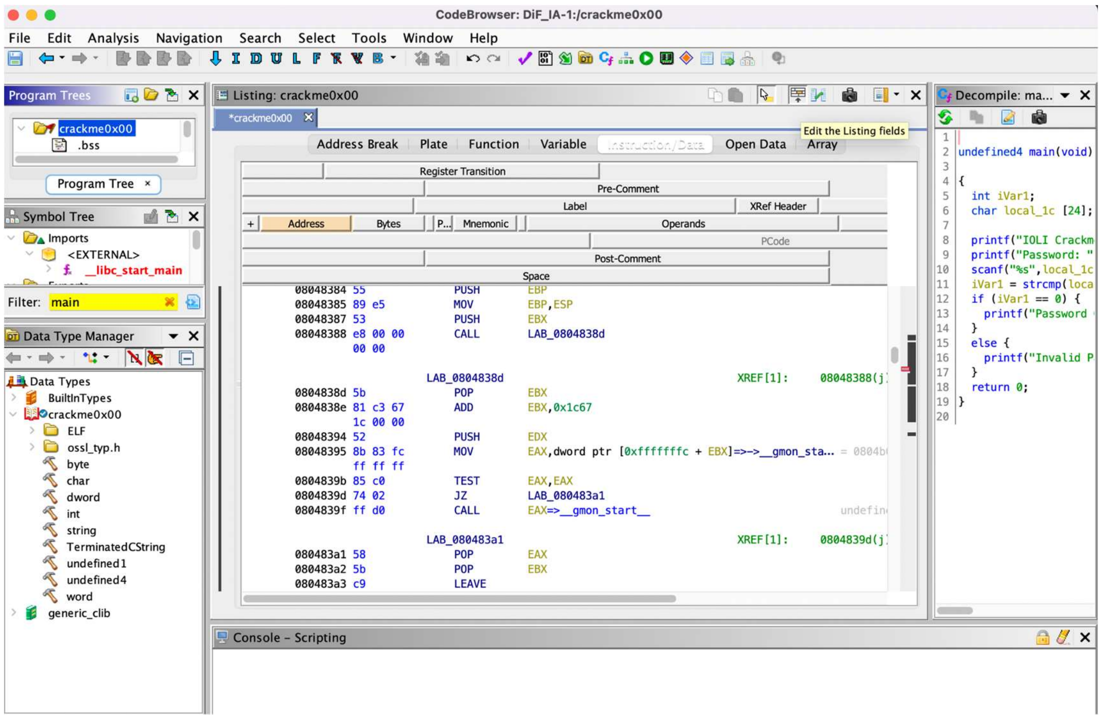
```
08048472 c7 04 24        MOV
         96 85 04 08
08048479 e8 c2 fe        CALL
         ff ff
0804847e eb 0c           JMP

                         LAB_0804
08048480 c7 04 24        MOV
         a9 85 04 08
08048487 e8 b4 fe        CALL
         ff ff

                         LAB_0804
0804848c b8 00 00        MOV
         00 00
08048491 c9              LEAV
08048492 c3              RET
08048493 90              ??
08048494 90              ??
08048495 90              ??
08048496 90              ??
08048497 90              ??
08048498 90              ??
08048499 90              ??
0804849a 90              ??
0804849b 90              ??
0804849c 90              ??
0804849d 90              ??
0804849e 90              ??
0804849f 90              ??
```

```c

undefined4 main(void)

{
  int iVar1;
  char local_1c [24];

  printf("IOLI Crackme Level 0x00\n");
  printf("Password: ");
  scanf("%s",local_1c);
  iVar1 = strcmp(local_1c,"250382");
  if (iVar1 == 0) {
    printf("Password OK :)\n");
  }
  else {
    printf("Invalid Password!\n");
  }
  return 0;
}
```


Uses of "char" (DataType) - 13 locations [CodeBrowser: DiF_IA-1:/crackme0x00]
Help
Uses of "char" (DataType) – 13 locations
Reference(s)
Filter:

Location	Label	Code Unit	Context
External[?]	scanf		[char * __format@Stack[0x4]:4]
External[?]	printf		[char * __format@Stack[0x4]:4]
External[?]	strcmp		[char * __s1@Stack[0x4]:4]
External[?]	strcmp		[char * __s2@Stack[0x8]:4]
08048414	main	PUSH EBP	6: char local_1c [24];
08048437		CALL printf	8: printf("IOLI Crackme Level 0x00\n");
08048443		CALL printf	9: printf("Password: ");
08048448		LEA EAX=>local_1c,[EBP + -0x18]	10: scanf("%s",local_1c);
08048456		CALL scanf	10: scanf("%s",local_1c);
0804845b		LEA EAX=>local_1c,[EBP + -0x18]	11: iVar1 = strcmp(local_1c,"250382");
08048469		CALL strcmp	11: iVar1 = strcmp(local_1c,"250382");
08048479		CALL printf	16: printf("Invalid Password!\n");
08048487		CALL printf	13: printf("Password OK :)\n");

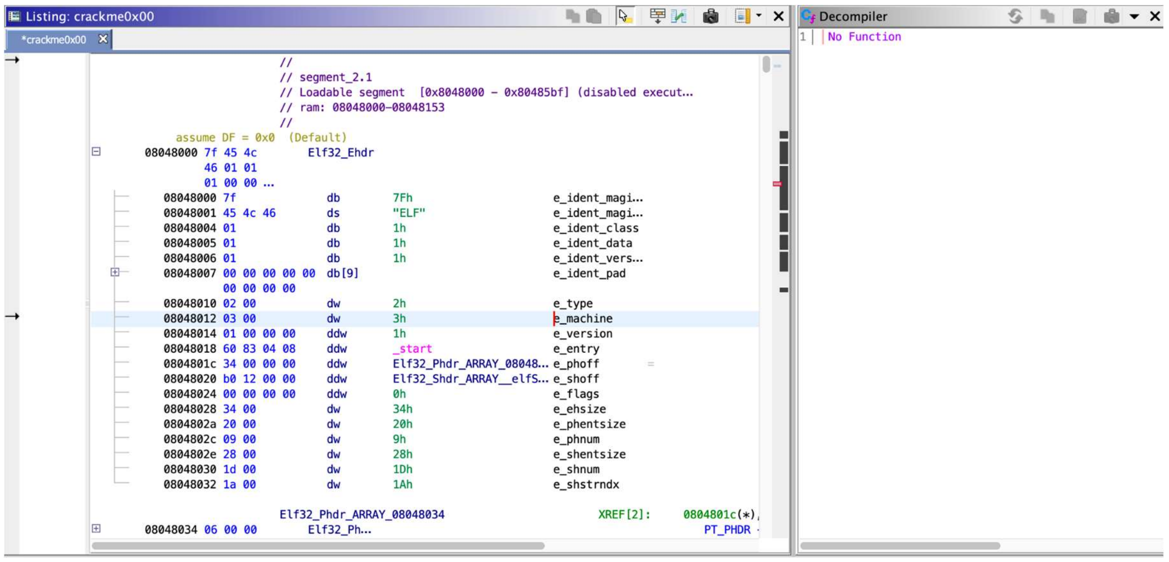
4. Listing Window: Displays the reverse-engineered code.
    a. Provides customization options by clicking “Edit the listing fields” in the upper right corner.



```
08048384 55          PUSH    EBP
08048385 89 e5       MOV     EBP,ESP
08048387 53          PUSH    EBX
08048388 e8 00 00    CALL    LAB_0804838d
         00 00

                 LAB_0804838d                              XREF[1]:     08048388(j
0804838d 5b          POP     EBX
0804838e 81 c3 67    ADD     EBX,0x1c67
         1c 00 00
08048394 52          PUSH    EDX
08048395 8b 83 fc    MOV     EAX,dword ptr [0xfffffffc + EBX]=>->__gmon_sta... = 0804b
         ff ff ff
0804839b 85 c0       TEST    EAX,EAX
0804839d 74 02       JZ      LAB_080483a1
0804839f ff d0       CALL    EAX=>__gmon_start__                          undefin

                 LAB_080483a1                              XREF[1]:     0804839d(j
080483a1 58          POP     EAX
080483a2 5b          POP     EBX
080483a3 c9          LEAVE
```

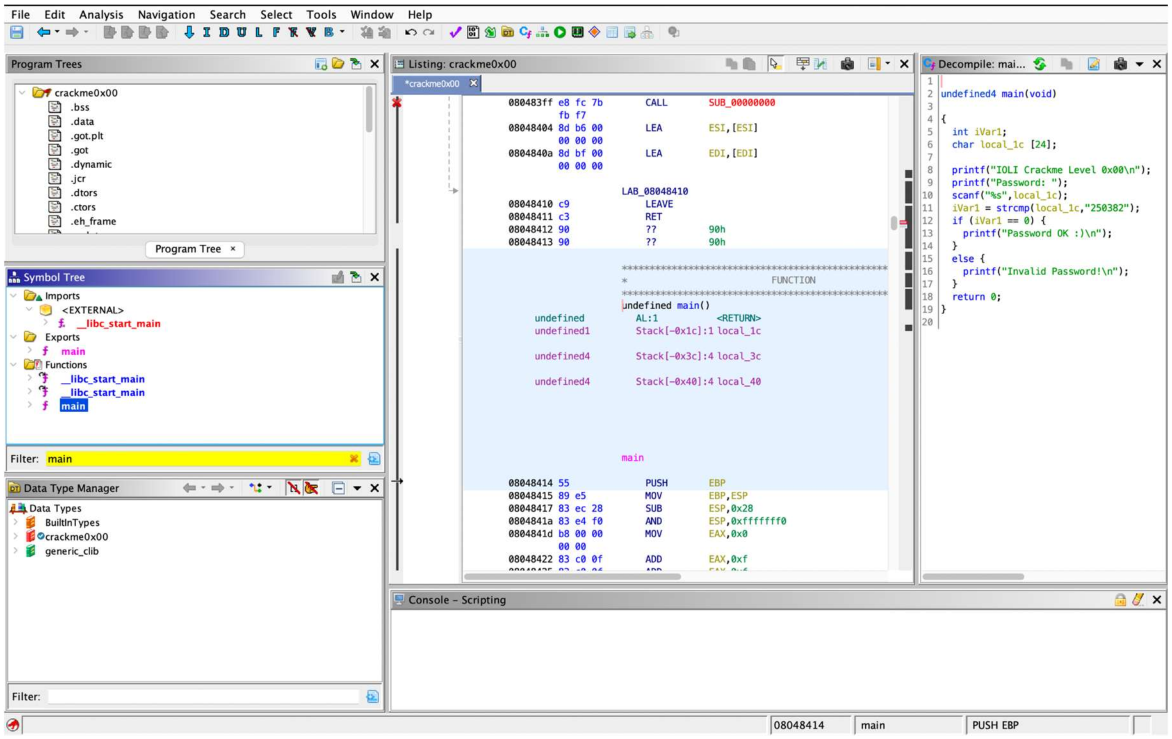
```

undefined4 main(void)

{
  int iVar1;
  char local_1c [24];

  printf("IOLI Crackm
  printf("Password: "
  scanf("%s",local_1c
  iVar1 = strcmp(loca
  if (iVar1 == 0) {
    printf("Password 
  }
  else {
    printf("Invalid P
  }
  return 0;
}

```

5. Decompiler: This window offers a high-level code representation.
    a. It correlates assembler instructions in the “Listing” window with high-level instructions. Highlighting a section in the Decompiler will also highlight the corresponding assembly section.



```
                          //
                          // segment_2.1
                          // Loadable segment  [0x8048000 - 0x80485bf] (disabled execut…
                          // ram: 08048000-08048153
                          //
              assume DF = 0x0  (Default)
     08048000 7f 45 4c        Elf32_Ehdr
              46 01 01
              01 00 00 …
        08048000 7f           db        7Fh                   e_ident_magi…
        08048001 45 4c 46     ds        "ELF"                 e_ident_magi…
        08048004 01           db        1h                    e_ident_class
        08048005 01           db        1h                    e_ident_data
        08048006 01           db        1h                    e_ident_vers…
        08048007 00 00 00 00 00 db[9]                         e_ident_pad
                 00 00 00 00
        08048010 02 00        dw        2h                    e_type
        08048012 03 00        dw        3h                    e_machine
        08048014 01 00 00 00  ddw       1h                    e_version
        08048018 60 83 04 08  ddw       _start                e_entry
        0804801c 34 00 00 00  ddw       Elf32_Phdr_ARRAY_08048… e_phoff
        08048020 b0 12 00 00  ddw       Elf32_Shdr_ARRAY__elfS… e_shoff
        08048024 00 00 00 00  ddw       0h                    e_flags
        08048028 34 00        dw        34h                   e_ehsize
        0804802a 20 00        dw        20h                   e_phentsize
        0804802c 09 00        dw        9h                    e_phnum
        0804802e 28 00        dw        28h                   e_shentsize
        08048030 1d 00        dw        1Dh                   e_shnum
        08048032 1a 00        dw        1Ah                   e_shstrndx

                          Elf32_Phdr_ARRAY_08048034                XREF[2]:     0804801c(*),
     08048034 06 00 00        Elf32_Ph…                                          PT_PHDR
```

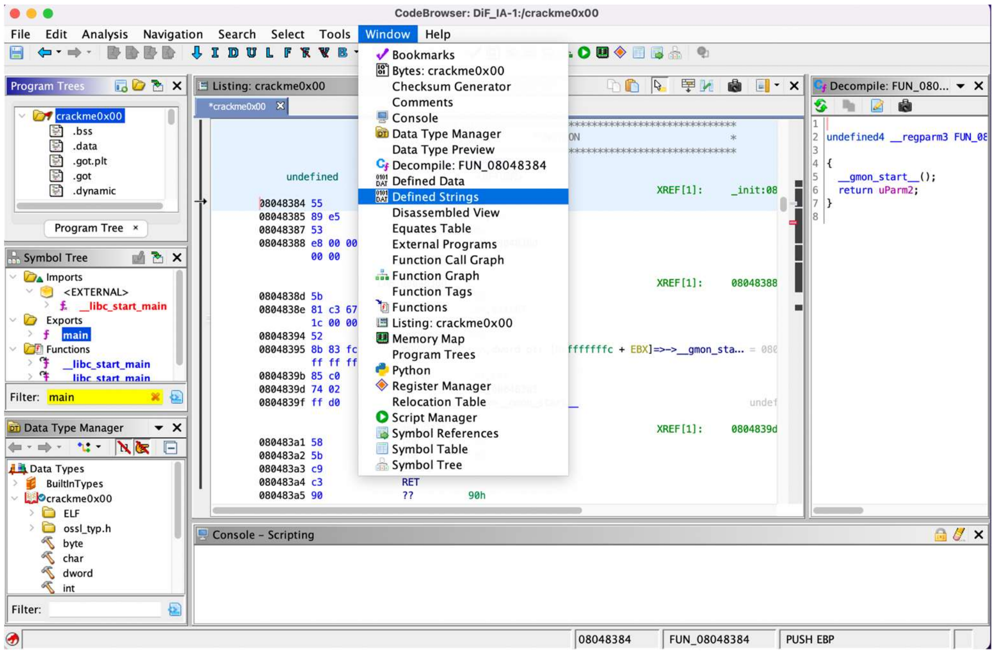
```
1 | No Function
```



```
080483ff e8 fc 7b      CALL      SUB_00000000
         fb f7
08048404 8d b6 00      LEA       ESI,[ESI]
         00 00 00
0804840a 8d bf 00      LEA       EDI,[EDI]
         00 00 00

                    LAB_08048410
08048410 c9            LEAVE
08048411 c3            RET
08048412 90            ??        90h
08048413 90            ??        90h

                    ****************************************************
                    *                         FUNCTION
                    ****************************************************
                    undefined main()
     undefined        AL:1           <RETURN>
     undefined1       Stack[-0x1c]:1 local_1c

     undefined4       Stack[-0x3c]:4 local_3c

     undefined4       Stack[-0x40]:4 local_40




                    main
08048414 55            PUSH      EBP
08048415 89 e5         MOV       EBP,ESP
08048417 83 ec 28      SUB       ESP,0x28
0804841a 83 e4 f0      AND       ESP,0xfffffff0
0804841d b8 00 00      MOV       EAX,0x0
         00 00
08048422 83 c0 0f      ADD       EAX,0xf
```

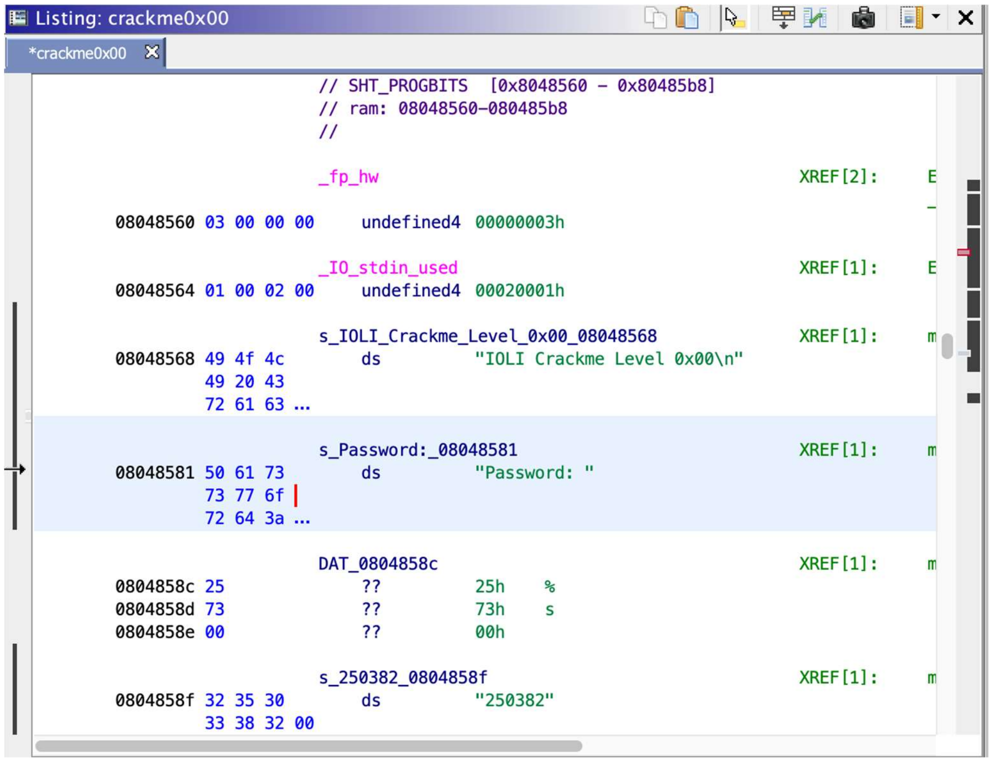
```c

undefined4 main(void)

{
  int iVar1;
  char local_1c [24];

  printf("IOLI Crackme Level 0x00\n");
  printf("Password: ");
  scanf("%s",local_1c);
  iVar1 = strcmp(local_1c,"250382");
  if (iVar1 == 0) {
    printf("Password OK :)\n");
  }
  else {
    printf("Invalid Password!\n");
  }
  return 0;
}

```

Step 12: Use Ghidra's “Window -> Defined Strings” for initial sorting of the binary file and to view program strings and identify potential clues or relevant parts of the code. Locate sections of the binary where user input (like a password) is processed.



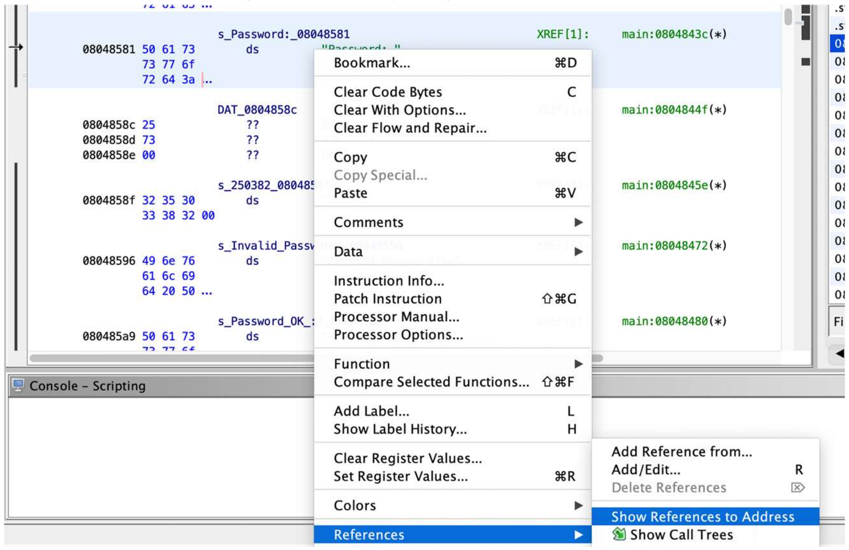
```
                 undefined
08048384 55
08048385 89 e5
08048387 53
08048388 e8 00 00
         00 00
0804838d 5b
0804838e 81 c3 67
         1c 00 00
08048394 52
08048395 8b 83 fc
         ff ff ff
0804839b 85 c0
0804839d 74 02
0804839f ff d0
080483a1 58
080483a2 5b
080483a3 c9
080483a4 c3                 RET
080483a5 90                 ??          90h
```

```
undefined4 __regparm3 FUN_08

{
  __gmon_start__();
  return uParm2;
}
```



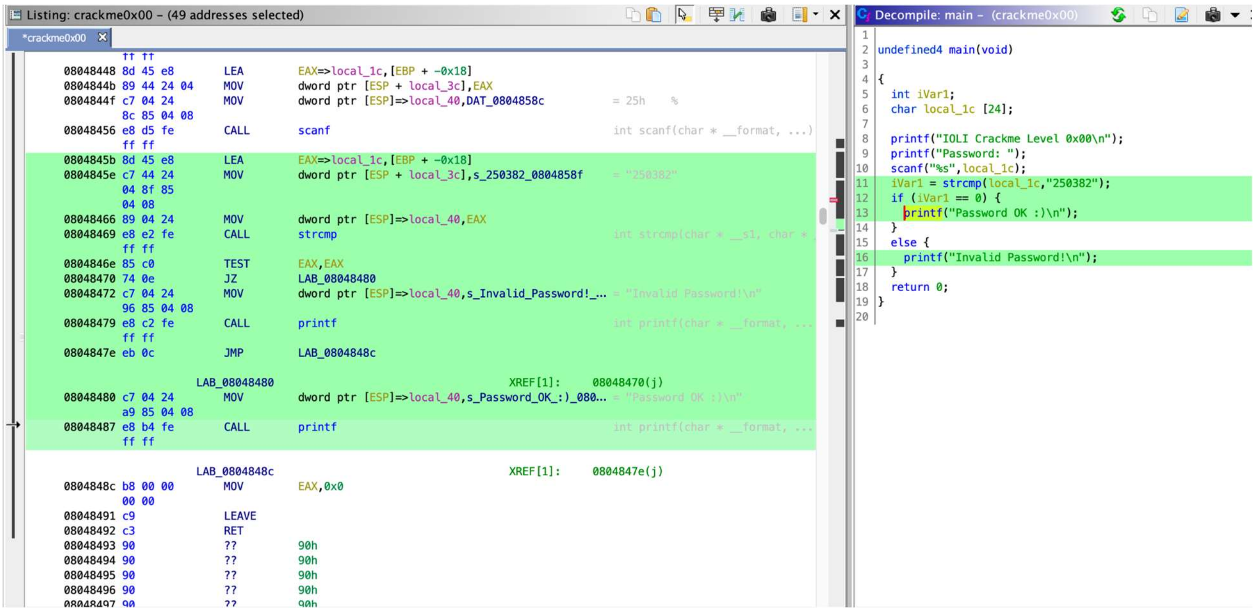
```
                            // SHT_PROGBITS  [0x8048560 - 0x80485b8]
                            // ram: 08048560-080485b8
                            //

                            _fp_hw                                      XREF[2]:     E

08048560 03 00 00 00     undefined4  00000003h

                            _IO_stdin_used                              XREF[1]:     E
08048564 01 00 02 00     undefined4  00020001h

                            s_IOLI_Crackme_Level_0x00_08048568          XREF[1]:     m
08048568 49 4f 4c        ds          "IOLI Crackme Level 0x00\n"
         49 20 43
         72 61 63 ...

                            s_Password:_08048581                        XREF[1]:     m
08048581 50 61 73        ds          "Password: "
         73 77 6f
         72 64 3a ...

                            DAT_0804858c                                XREF[1]:     m
0804858c 25              ??          25h    %
0804858d 73              ??          73h    s
0804858e 00              ??          00h

                            s_250382_0804858f                           XREF[1]:     m
0804858f 32 35 30        ds          "250382"
         33 38 32 00
```

Step 13: Left-click the address and select “References -> Show References to Address”. Another alternative is clicking on the entry to go to the code section with “Password” reference.



```
                     s_Password:_08048581                      XREF[1]:     main:0804843c(*)
08048581 50 61 73        ds
         73 77 6f
         72 64 3a ...

                     DAT_0804858c                                            main:0804844f(*)
0804858c 25              ??
0804858d 73              ??
0804858e 00              ??

                     s_250382_08048                                          main:0804845e(*)
0804858f 32 35 30        ds
         33 38 32 00

                     s_Invalid_Passw                                         main:08048472(*)
08048596 49 6e 76        ds
         61 6c 69
         64 20 50 ...

                     s_Password_OK_:                                         main:08048480(*)
080485a9 50 61 73        ds
```

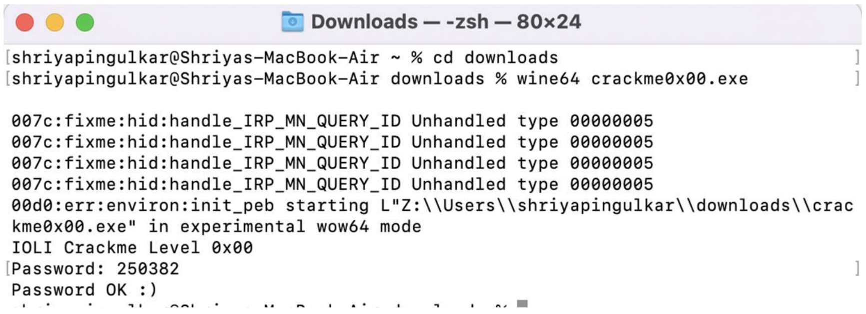
Step 14: Upon encountering a reference to "Password" in the binary's code, a user input is captured via the `scanf` function. This input is stored in the EAX register and subsequently saved in the local variable “local_40”. A hardcoded password "250382" is also saved in another local variable “local_3c”. Both strings (user input and hardcoded password) are compared using the `strcmp` function. If they match (i.e., `strcmp` returns zero), the program outputs "Password OK :)", otherwise it displays "Invalid Password!".


Listing: crackme0x00 – (49 addresses selected)
*crackme0x00
Decompile: main – (crackme0x00)

```
08048448 8d 45 e8        LEA        EAX=>local_1c,[EBP + -0x18]
0804844b 89 44 24 04     MOV        dword ptr [ESP + local_3c],EAX
0804844f c7 04 24        MOV        dword ptr [ESP]=>local_40,DAT_0804858c          = 25h    %
         8c 85 04 08
08048456 e8 d5 fe        CALL       scanf                                           int scanf(char * __format, ...)
         ff ff
0804845b 8d 45 e8        LEA        EAX=>local_1c,[EBP + -0x18]
0804845e c7 44 24        MOV        dword ptr [ESP + local_3c],s_250382_0804858f    = "250382"
         04 8f 85
         04 08
08048466 89 04 24        MOV        dword ptr [ESP]=>local_40,EAX
08048469 e8 e2 fe        CALL       strcmp                                          int strcmp(char * __s1, char *
         ff ff
0804846e 85 c0           TEST       EAX,EAX
08048470 74 0e           JZ         LAB_08048480
08048472 c7 04 24        MOV        dword ptr [ESP]=>local_40,s_Invalid_Password!_... = "Invalid Password!\n"
         96 85 04 08
08048479 e8 c2 fe        CALL       printf                                          int printf(char * __format, ...
         ff ff
0804847e eb 0c           JMP        LAB_0804848c

                     LAB_08048480                              XREF[1]:     08048470(j)
08048480 c7 04 24        MOV        dword ptr [ESP]=>local_40,s_Password_OK_:)_080... = "Password OK :)\n"
         a9 85 04 08
08048487 e8 b4 fe        CALL       printf                                          int printf(char * __format, ...
         ff ff

                     LAB_0804848c                              XREF[1]:     0804847e(j)
0804848c b8 00 00        MOV        EAX,0x0
         00 00
08048491 c9              LEAVE
08048492 c3              RET
08048493 90              ??         90h
08048494 90              ??         90h
08048495 90              ??         90h
08048496 90              ??         90h
08048497 90              ??         90h
```

```

undefined4 main(void)

{
  int iVar1;
  char local_1c [24];

  printf("IOLI Crackme Level 0x00\n");
  printf("Password: ");
  scanf("%s",local_1c);
  iVar1 = strcmp(local_1c,"250382");
  if (iVar1 == 0) {
    printf("Password OK :)\n");
  }
  else {
    printf("Invalid Password!\n");
  }
  return 0;
}

```

Step 15: Run "crackme0x00.exe" again, assign it the password "250382".



```
shriyapingulkar@Shriyas-MacBook-Air ~ % cd downloads
shriyapingulkar@Shriyas-MacBook-Air downloads % wine64 crackme0x00.exe

007c:fixme:hid:handle_IRP_MN_QUERY_ID Unhandled type 00000005
007c:fixme:hid:handle_IRP_MN_QUERY_ID Unhandled type 00000005
007c:fixme:hid:handle_IRP_MN_QUERY_ID Unhandled type 00000005
007c:fixme:hid:handle_IRP_MN_QUERY_ID Unhandled type 00000005
00d0:err:environ:init_peb starting L"Z:\\Users\\shriyapingulkar\\downloads\\crac
kme0x00.exe" in experimental wow64 mode
IOLI Crackme Level 0x00
Password: 250382
Password OK :)
```