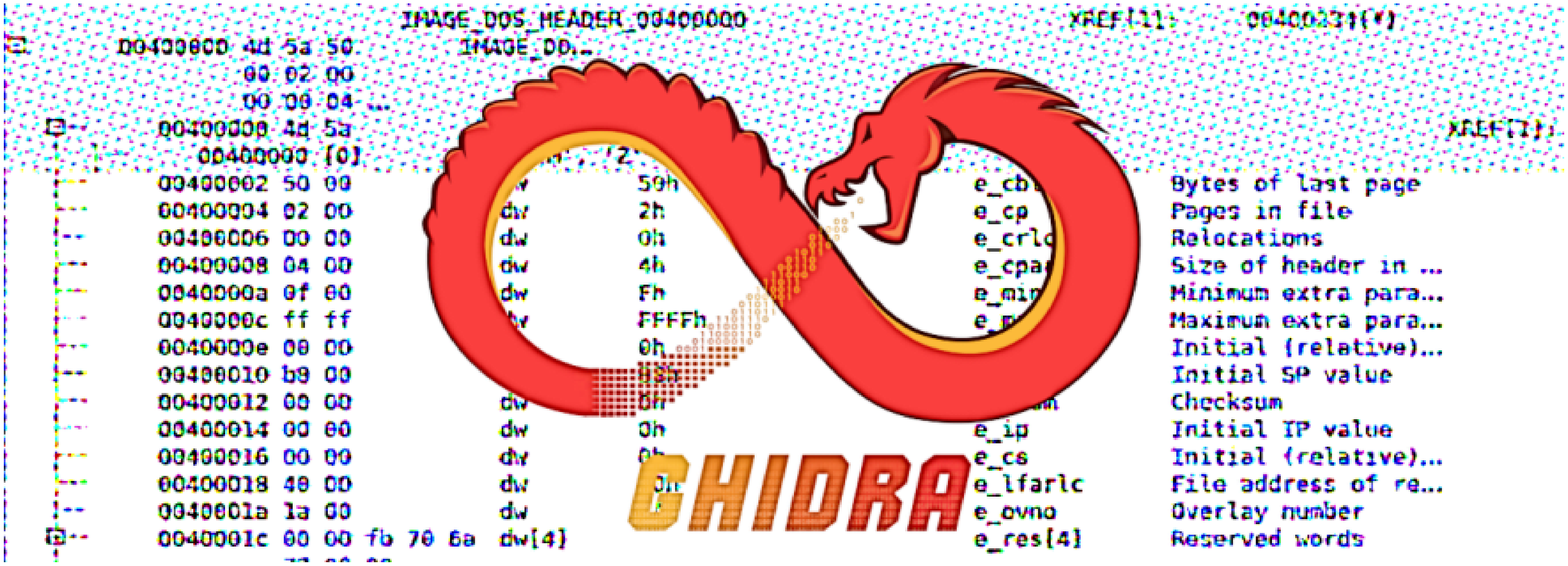
Getting Started with Reverse Engineering using Ghidra

Hi Peerlysters

In this article, we are going to explore how to download Ghidra, install it and use it to perform many important tasks such as reverse engineering, binary analysis and malware analysis.

To get the most from this article I attached a helpful doc that contains many useful links to learn reverse engineering and assembly: Reverse Engineering Resources



Source

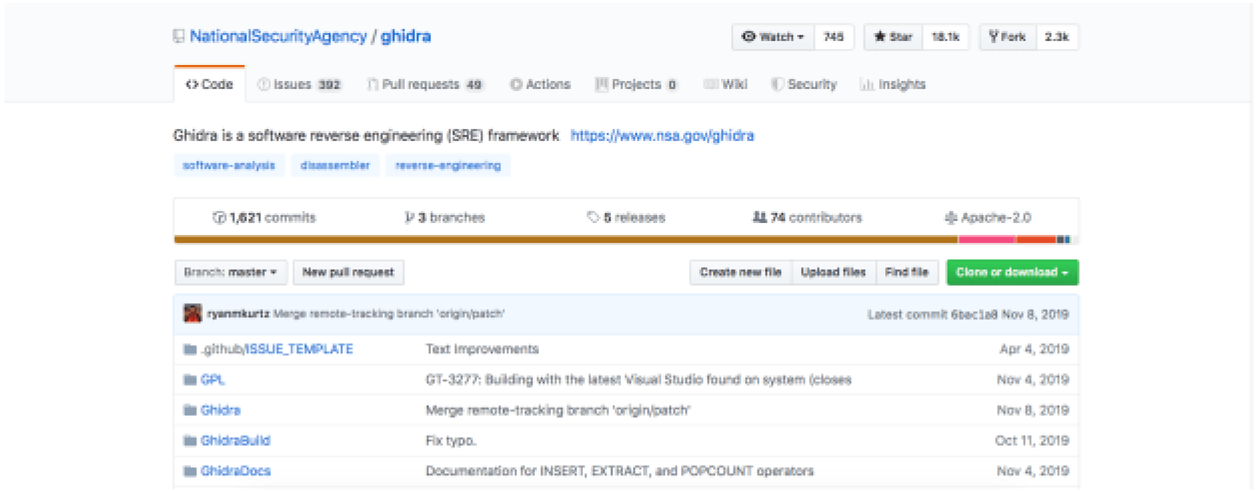
But first what is Ghidra exactly?

According to its official Github repository:

“Ghidra is a software reverse engineering (SRE) framework created and maintained by the National Security AgencyResearch Directorate. This framework includes a suite of full-featured, high-end software analysis tools that enable users to analyze compiled code on a variety of platforms including Windows, macOS, and Linux. Capabilities include disassembly, assembly, decompilation, graphing, and scripting, along with hundreds of other features. Ghidra supports a wide variety of processorinstruction sets and executable formats and can be run in both user-interactive and automated modes. Users may also develop their own Ghidra plug-in components and/or scripts using Java or Python.

In support of NSA's Cyber Security mission, Ghidra was built to solve scaling and teaming problems on complex SRE efforts, and to provide a customizable and extensible SRE research platform. NSA has applied Ghidra SRE capabilities to a variety of problems that involve analyzing malicious code and generating deep insights for SRE analysts who seek a better understanding of potential vulnerabilities in networks and systems.

https://github.com/NationalSecurityAgency/ghidra



NationalSecurityAgency / ghidra

Ghidra is a software reverse engineering (SRE) framework https://www.nsa.gov/ghidra

1,621 commits | 3 branches | 5 releases | 74 contributors | Apache-2.0

ryanmkurtz Merge remote-tracking branch 'origin/patch' — Latest commit 6bac1a8 Nov 8, 2019

.github/ISSUE_TEMPLATE	Text improvements	Apr 4, 2019
GPL	GT-3277: Building with the latest Visual Studio found on system (closes	Nov 4, 2019
Ghidra	Merge remote-tracking branch 'origin/patch'	Nov 8, 2019
GhidraBuild	Fix typo.	Oct 11, 2019
GhidraDocs	Documentation for INSERT, EXTRACT, and POPCOUNT operators	Nov 4, 2019

The official website of the project is https://ghidra-sre.org:

As you can notice from the official description that this tool was developed and maintained by the US NSA (National Security Agency) which leads us to think about if this tool is secure. Check this post if you didn't know what i am talking about:

Compilation example with a C Program:

Before diving into the fundamentals of reverse engineering with this powerful tool (Ghidra) , let's explore the compiling phases in order to get an executable and some important terminologies.

Wikipedia defines Reverse engineering as follows:

> *"Reverse engineering, also called back engineering, is the process by which a human-made object is deconstructed to reveal its designs, architecture, or to extract knowledge from the object; similar to scientific research, the only difference being that scientific research is about a natural phenomenon."*

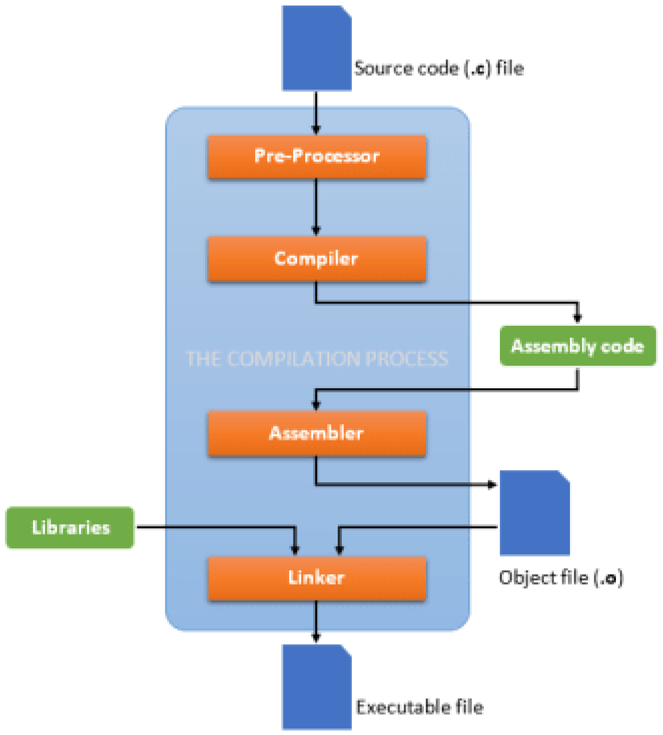
Compilers: convert high-level code to assembly code

Assemblers: convert assembly code to machine code

Linkers: take the object files in order to generate the executable

Disassemblers: convert machine code to assembly code

The phases are represented in the following graph:



Figure

As a demonstration, let’s compile a simple c program. The most known easy program is simply a “**hello world!**” program

Create a **hello.c** program:

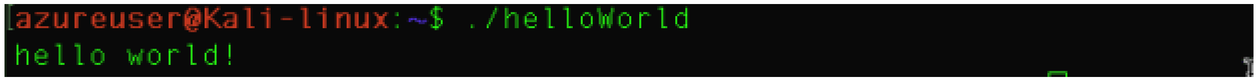
```
#include <stdio.h>

void main(void)

{
    printf ("hello world!\n");

}
```

Now let’s compile it and link it with gcc

```
gcc -o helloWorld hello.c
```

Run the executable

```
./helloWorld
```



```
azureuser@Kali-linux:~$ ./helloWorld
hello world!
```

How to install Ghidra?

To use Ghidra we need to install it of course. As technical requirements, you need the following

Hardware

- 4 GB RAM
- 1 GB storage (for installed Ghidra binaries)
- Dual monitors strongly suggested

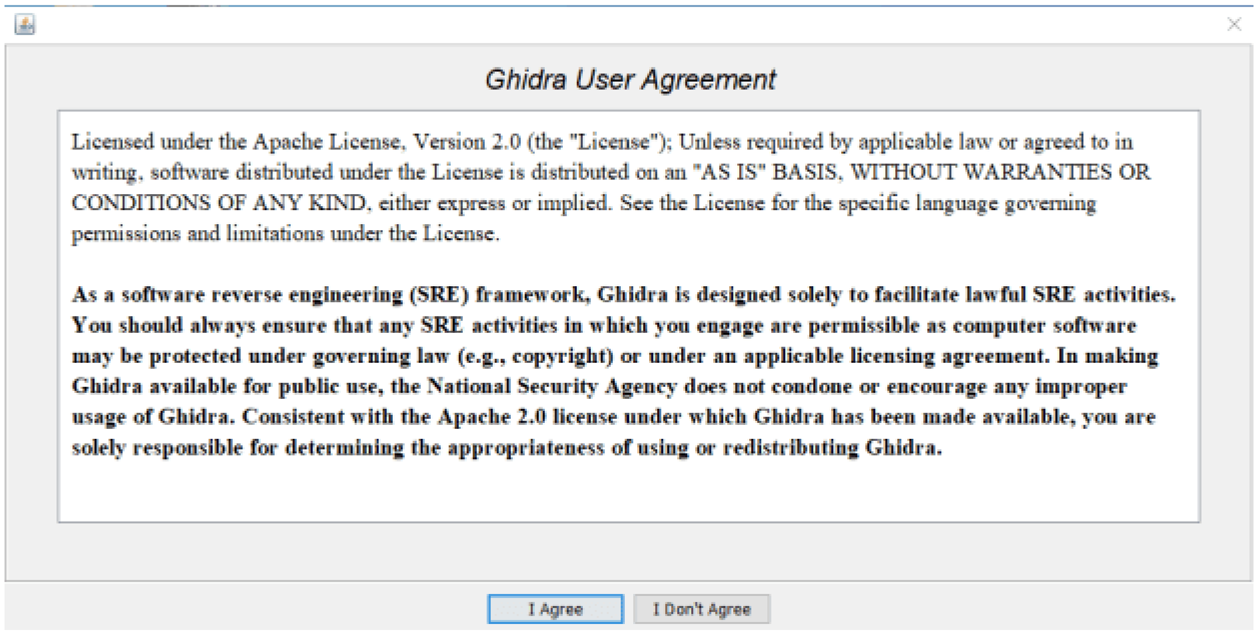
Software

- Java 11 64-bit Runtime and Development Kit (JDK)

Go to [Download Ghidra v9.1]

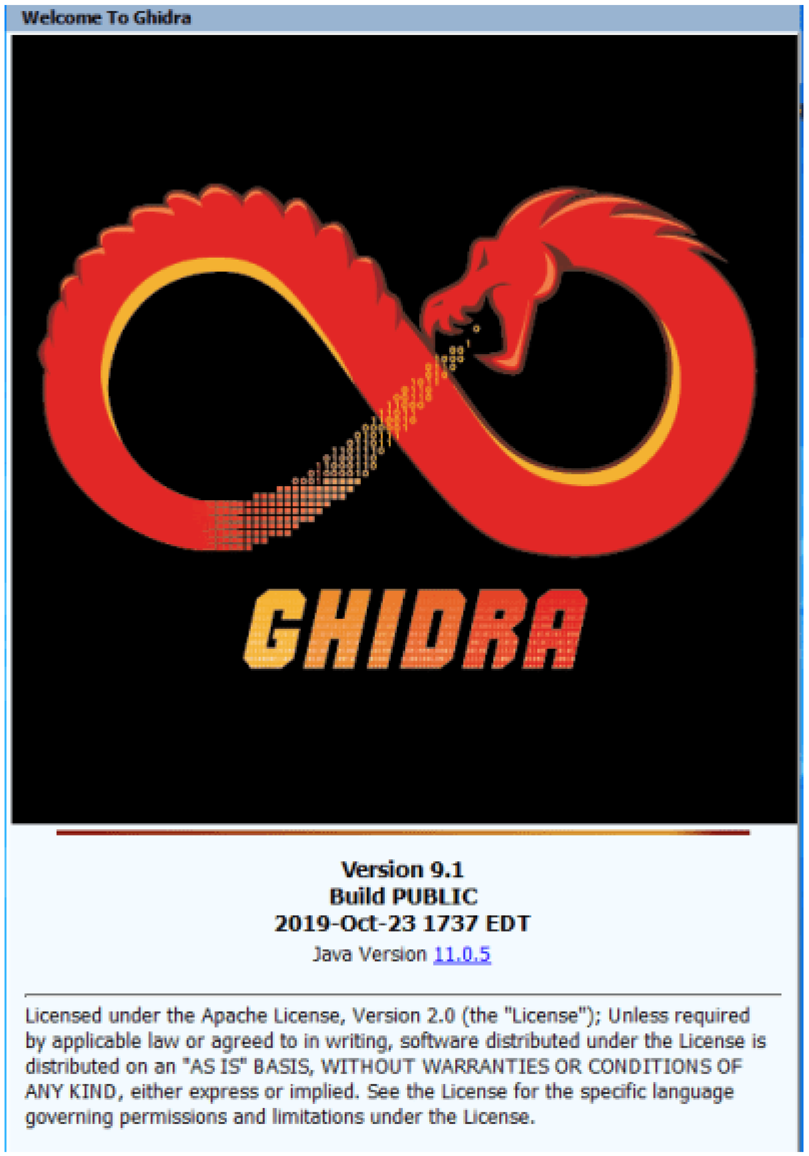
Download it and install Java JDK

Go to the installation folder and run the Ghidra bat file



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.


Welcome To Ghidra

Version 9.1
Build PUBLIC
2019-Oct-23 1737 EDT
Java Version 11.0.5

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.

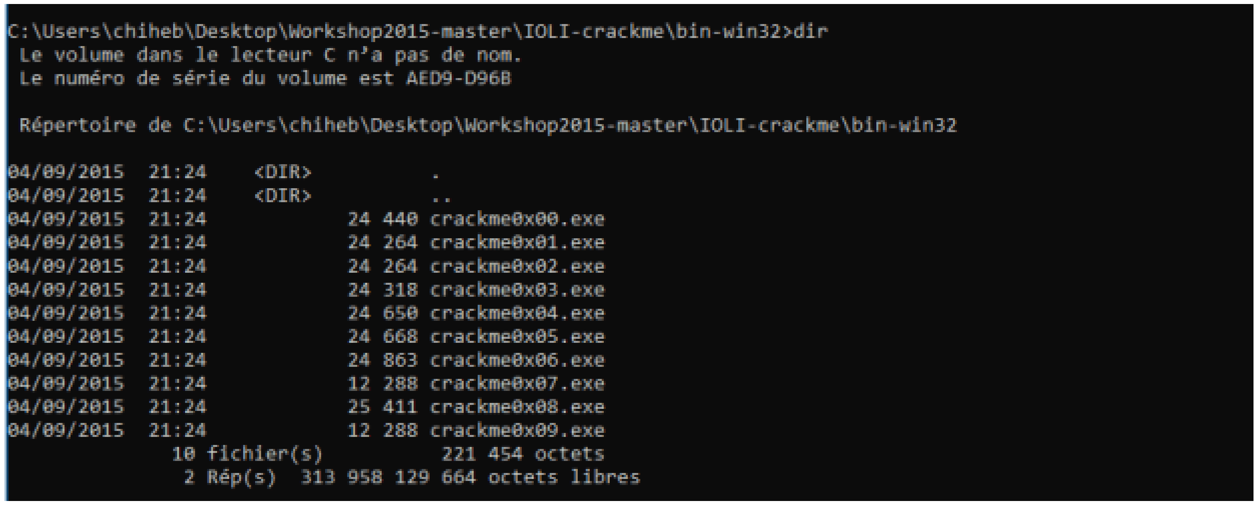
For more information about the installation steps you can check Ghidra official documentation: https://ghidra-sre.org/InstallationGuide.html

Reverse engineering example (CrackMe Challenge):

We learned the compilation phases in order to generate a fully working binary. Now it is time to continue our learning experiencewith acquiring some fundamentals about reverse engineering. That is why we are going to download a small and easy CrackMe challenge and we will try to understand what is doing and how it works in order to find the correct password to solve the challenges.

The challenge that we are going to solve is a part of this free and publicly available training materials: https://github.com/Maijin/Workshop2015

Download the GitHub repository, go to /IOLI-crackme/bin-win32 and you will find the challenge binaries.



```
C:\Users\chiheb\Desktop\Workshop2015-master\IOLI-crackme\bin-win32>dir
 Le volume dans le lecteur C n'a pas de nom.
 Le numéro de série du volume est AED9-D96B

 Répertoire de C:\Users\chiheb\Desktop\Workshop2015-master\IOLI-crackme\bin-win32

04/09/2015  21:24    <DIR>          .
04/09/2015  21:24    <DIR>          ..
04/09/2015  21:24            24 440 crackme0x00.exe
04/09/2015  21:24            24 264 crackme0x01.exe
04/09/2015  21:24            24 264 crackme0x02.exe
04/09/2015  21:24            24 318 crackme0x03.exe
04/09/2015  21:24            24 650 crackme0x04.exe
04/09/2015  21:24            24 668 crackme0x05.exe
04/09/2015  21:24            24 863 crackme0x06.exe
04/09/2015  21:24            12 288 crackme0x07.exe
04/09/2015  21:24            25 411 crackme0x08.exe
04/09/2015  21:24            12 288 crackme0x09.exe
              10 fichier(s)          221 454 octets
               2 Rép(s)  313 958 129 664 octets libres
```

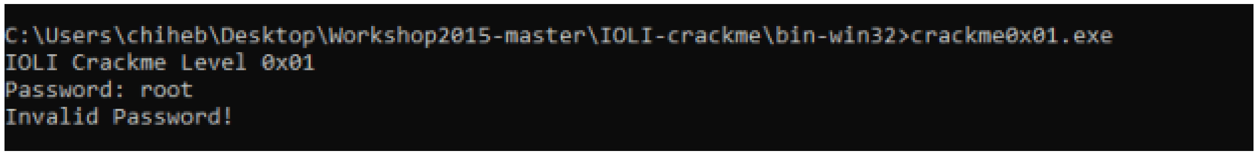
We are going to reverse "**Crackme0x01**" file.

Let's open it directly using the command line terminal:

Enter the binaries folder and type:

Crackme0x01.exe

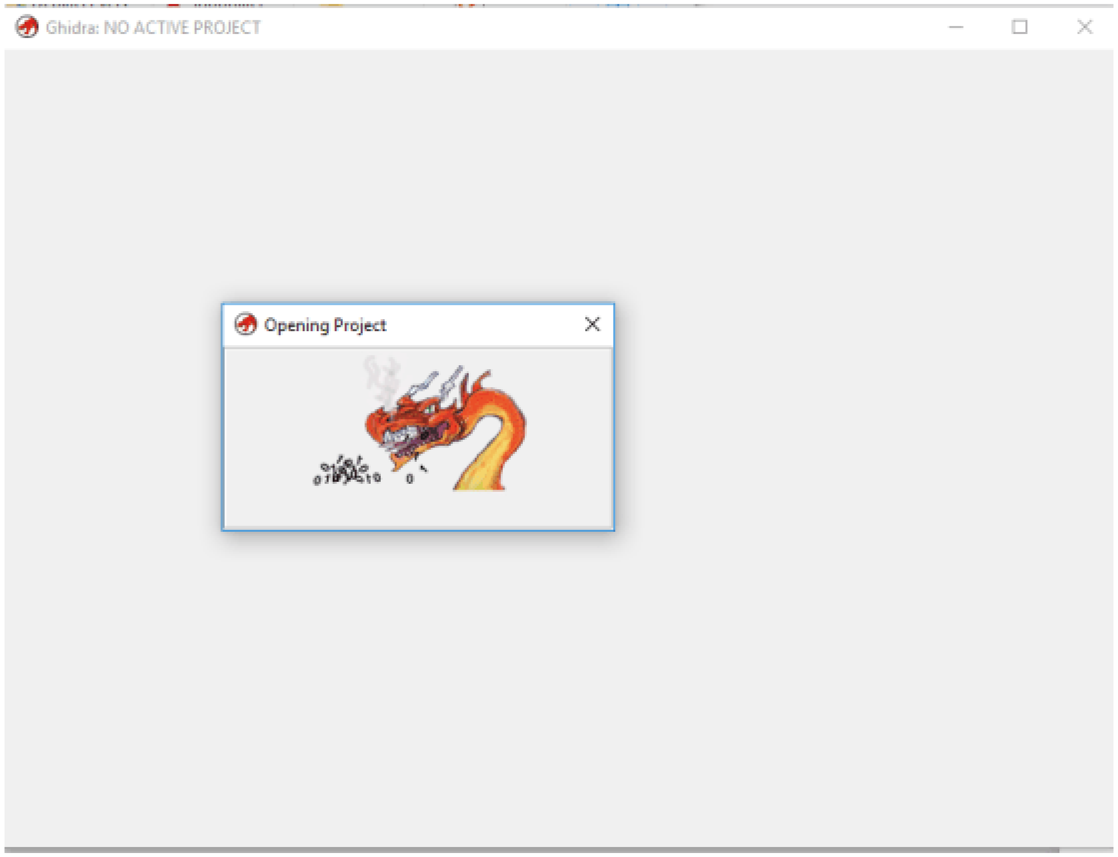
Enter a random password. In my case I entered "root" but i get an "Invalid Password!" error message



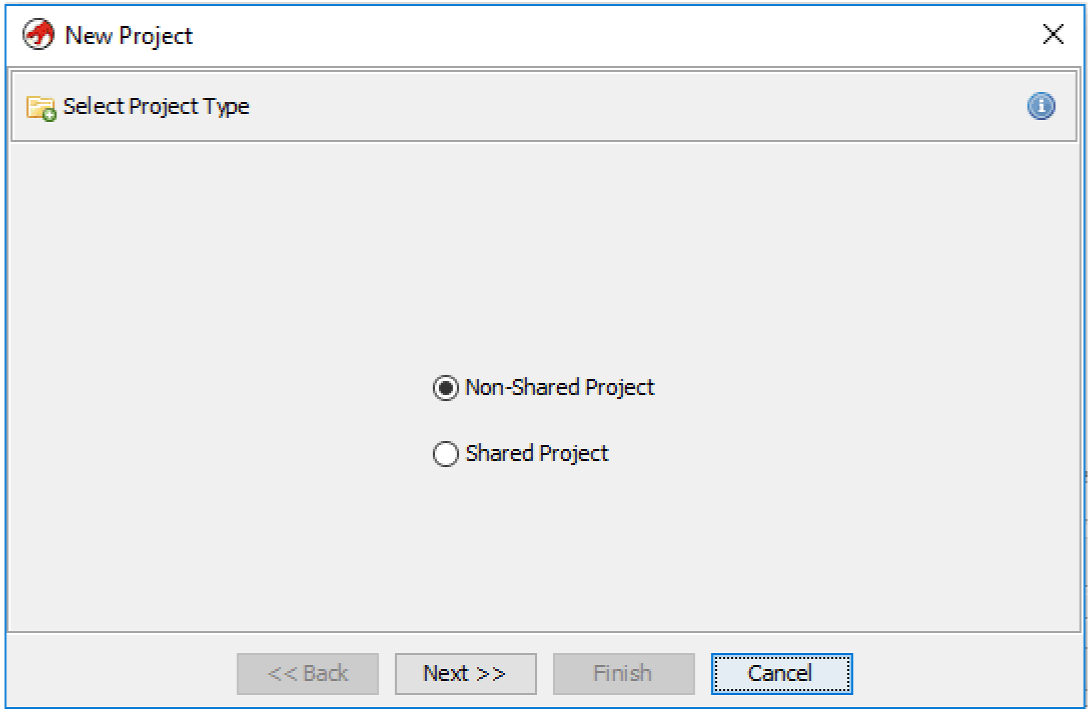
```
C:\Users\chiheb\Desktop\Workshop2015-master\IOLI-crackme\bin-win32>crackme0x01.exe
IOLI Crackme Level 0x01
Password: root
Invalid Password!
```

Then let’s crack it

Open Ghidra



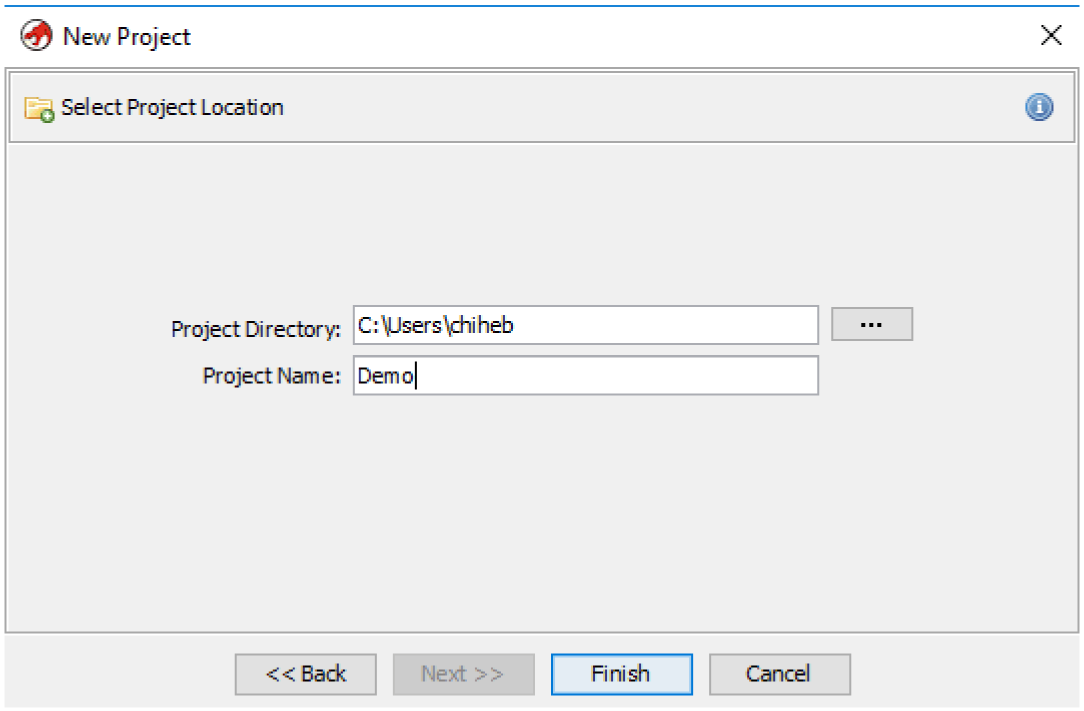
Start a new project:



● Non-Shared Project

○ Shared Project

Name the project

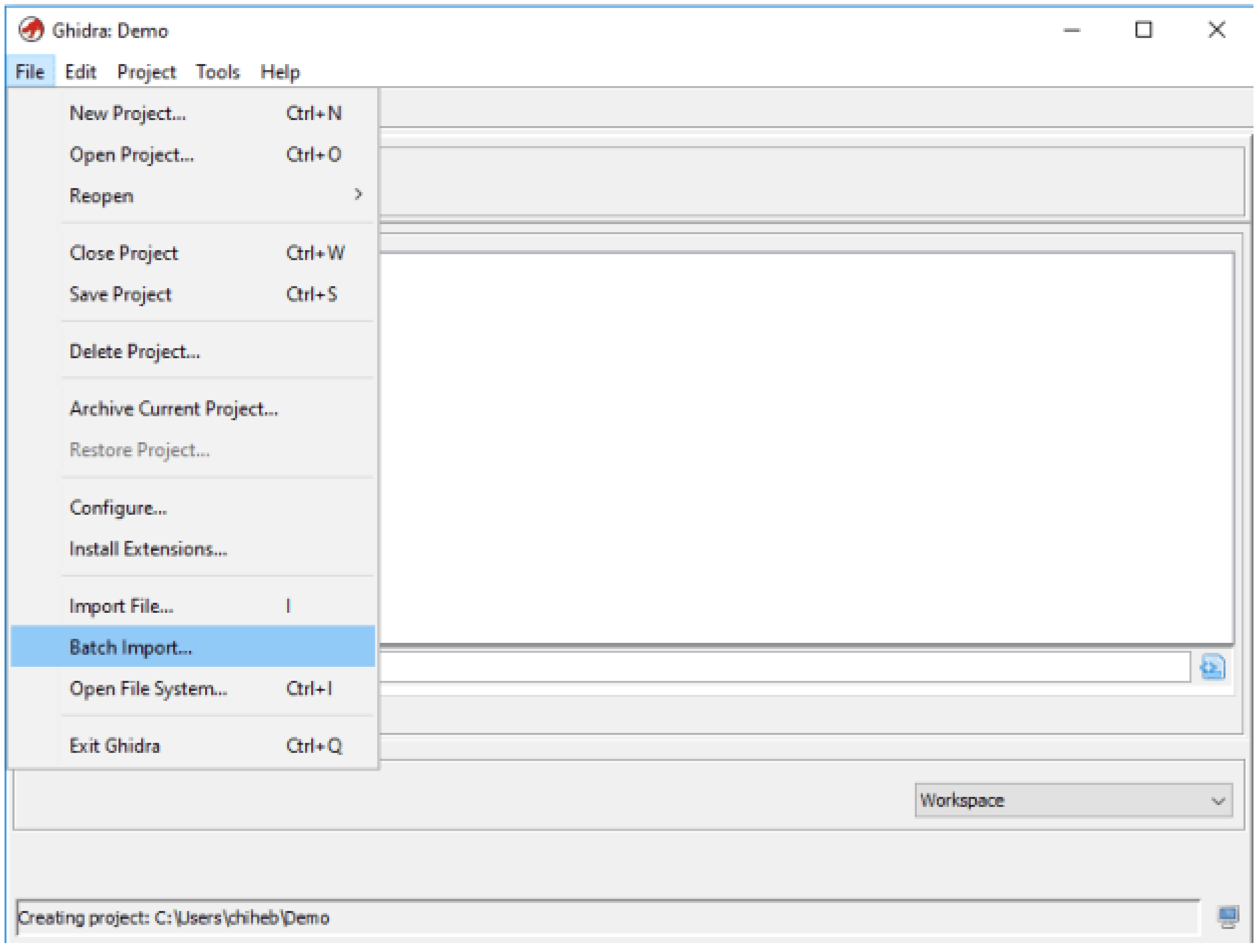

New Project

Select Project Location

Project Directory: C:\Users\chiheb

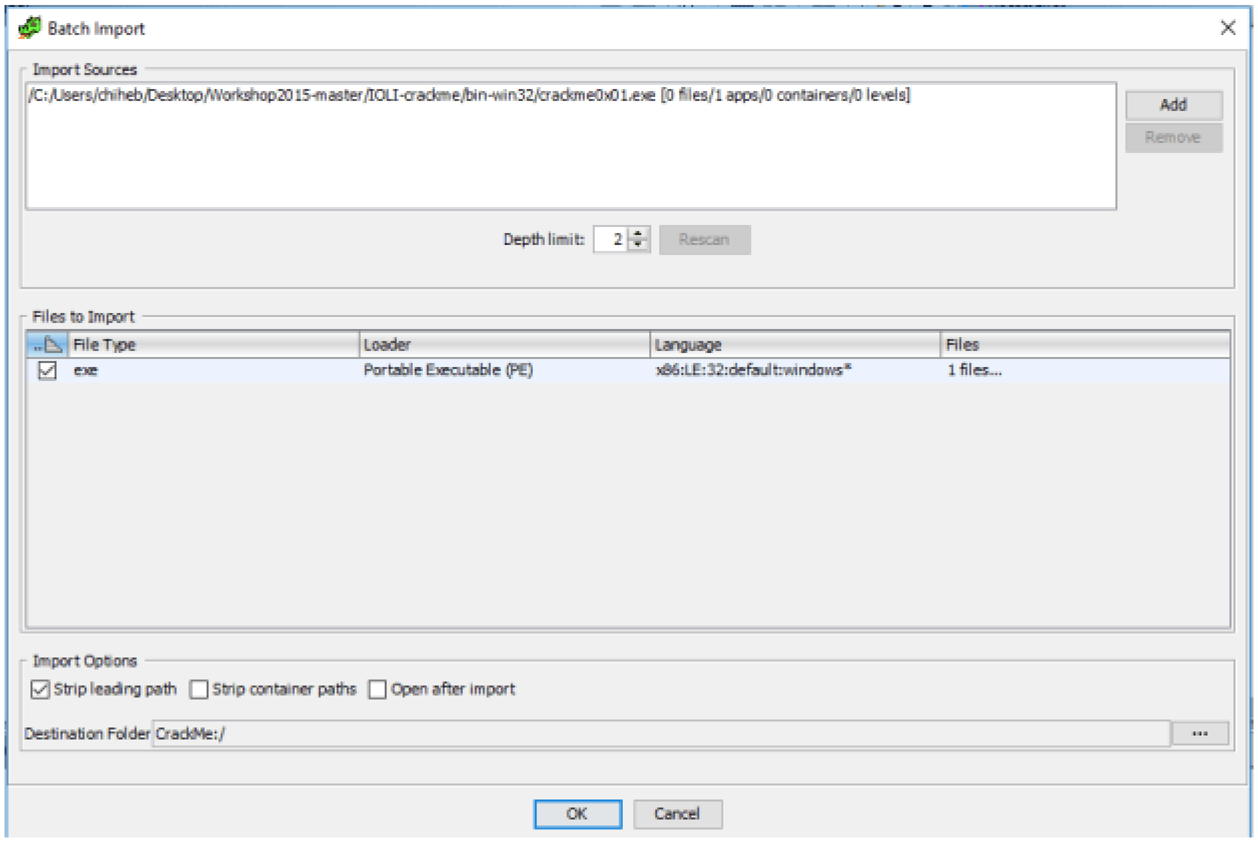
Project Name: Demo

Import the binary with **Batch Import**



Creating project: C:\Users\chiheb\Demo

Open the binary



/C:/Users/chiheb/Desktop/Workshop2015-master/IOLI-crackme/bin-win32/crackme0x01.exe [0 files/1 apps/0 containers/0 levels]

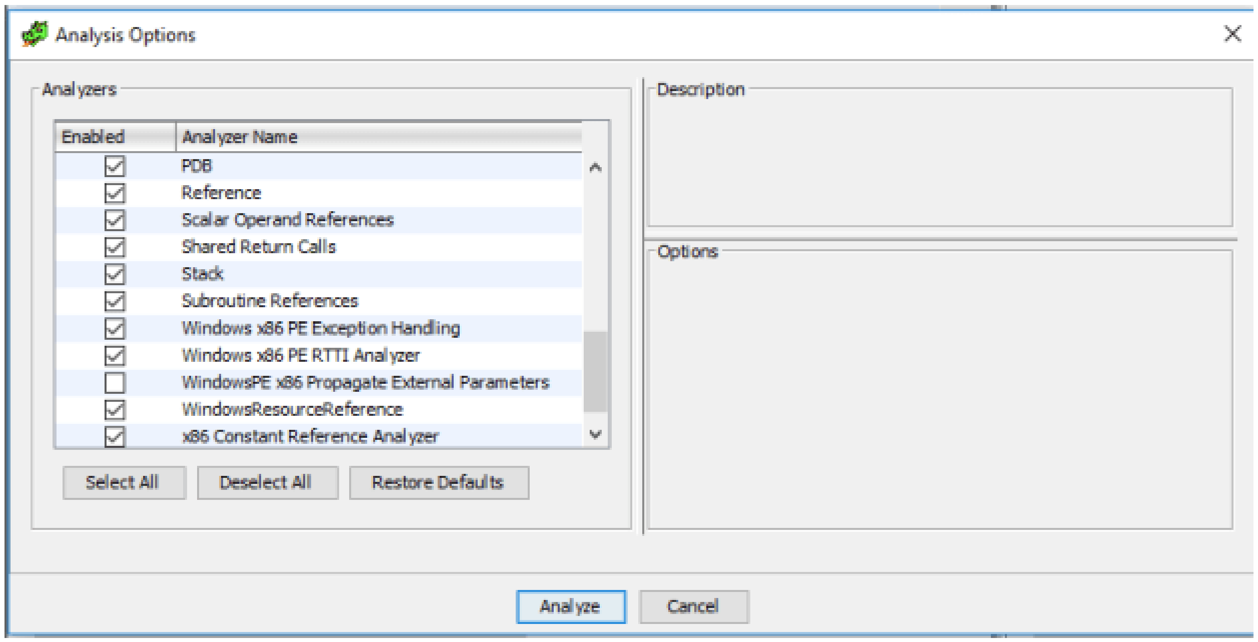
Depth limit: 2

	File Type	Loader	Language	Files
✓	exe	Portable Executable (PE)	x86:LE:32:default:windows*	1 files...

[✓] Strip leading path [ ] Strip container paths [ ] Open after import

Destination Folder CrackMe:/

Select the required options and click “Analyze”

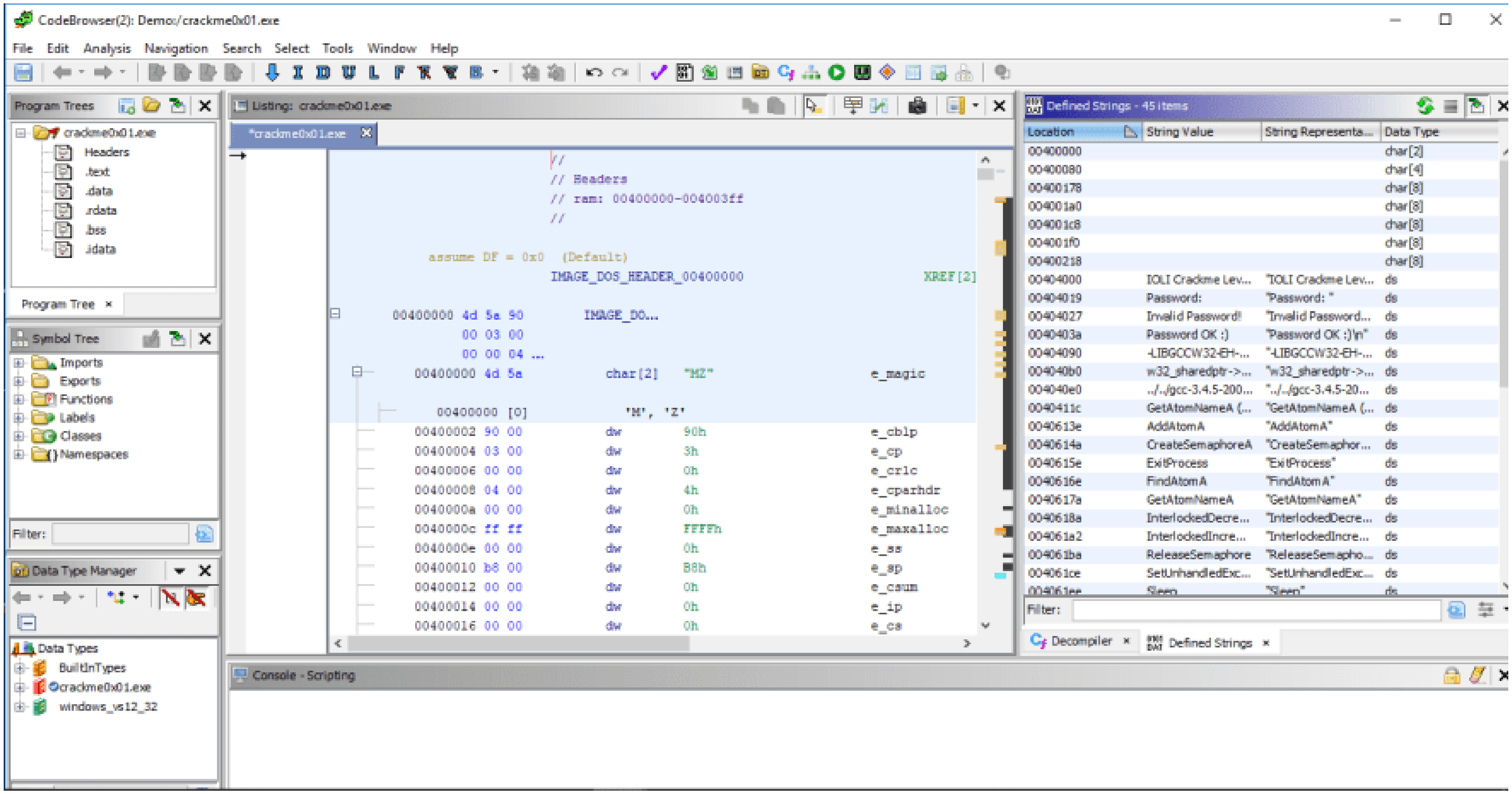


Analysis Options

Analyzers

Enabled	Analyzer Name
✓	PDB
✓	Reference
✓	Scalar Operand References
✓	Shared Return Calls
✓	Stack
✓	Subroutine References
✓	Windows x86 PE Exception Handling
✓	Windows x86 PE RTTI Analyzer
☐	WindowsPE x86 Propagate External Parameters
✓	WindowsResourceReference
✓	x86 Constant Reference Analyzer

Voila! This is the main windows of Ghidra

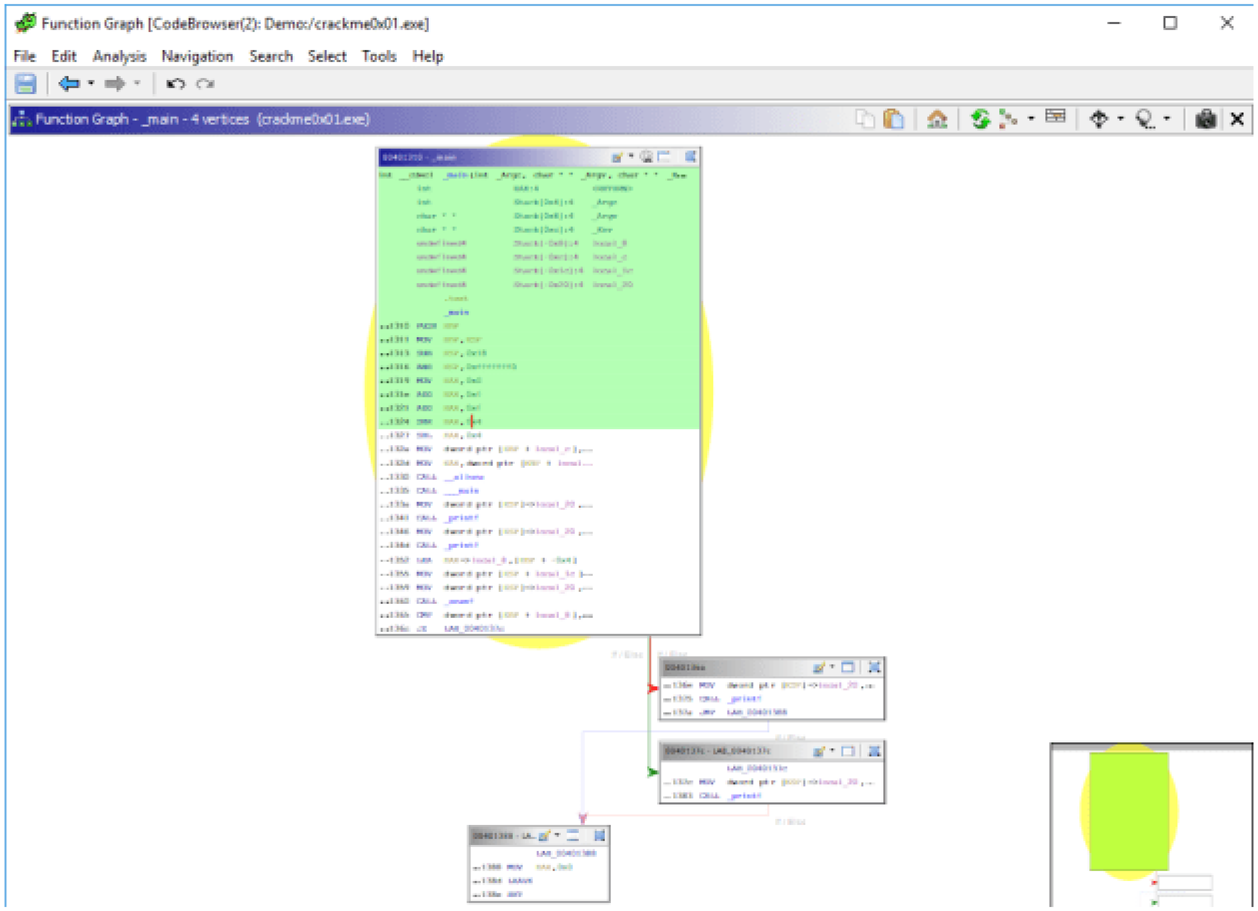


```
                         //
                         // Headers
                         // ram: 00400000-004003ff
                         //

           assume DF = 0x0  (Default)
                         IMAGE_DOS_HEADER_00400000                        XREF[2]

     00400000 4d 5a 90         IMAGE_DO...
              00 03 00
              00 00 04 ...
        00400000 4d 5a             char[2]    "MZ"                e_magic

          00400000 [0]               'M', 'Z'
        00400002 90 00             dw         90h                 e_cblp
        00400004 03 00             dw         3h                  e_cp
        00400006 00 00             dw         0h                  e_crlc
        00400008 04 00             dw         4h                  e_cparhdr
        0040000a 00 00             dw         0h                  e_minalloc
        0040000c ff ff             dw         FFFFh               e_maxalloc
        0040000e 00 00             dw         0h                  e_ss
        00400010 b8 00             dw         B8h                 e_sp
        00400012 00 00             dw         0h                  e_csum
        00400014 00 00             dw         0h                  e_ip
        00400016 00 00             dw         0h                  e_cs
```

Defined Strings - 45 items

Location	String Value	String Representa...	Data Type
00400000			char[2]
00400080			char[4]
00400178			char[8]
004001a0			char[8]
004001c8			char[8]
004001f0			char[8]
00400218			char[8]
00404000	IOLI Crackme Lev...	"IOLI Crackme Lev...	ds
00404019	Password:	"Password: "	ds
00404027	Invalid Password!	"Invalid Password...	ds
0040403a	Password OK :)	"Password OK :)\n"	ds
00404090	-LIBGCCW32-EH-...	"-LIBGCCW32-EH-...	ds
004040b0	w32_sharedptr->...	"w32_sharedptr->...	ds
004040e0	../../gcc-3.4.5-200...	"../../gcc-3.4.5-20...	ds
0040411c	GetAtomNameA (...	"GetAtomNameA (...	ds
0040613e	AddAtomA	"AddAtomA"	ds
0040614a	CreateSemaphoreA	"CreateSemaphor...	ds
0040615e	ExitProcess	"ExitProcess"	ds
0040616e	FindAtomA	"FindAtomA"	ds
0040617a	GetAtomNameA	"GetAtomNameA"	ds
0040618a	InterlockedDecre...	"InterlockedDecre...	ds
004061a2	InterlockedIncre...	"InterlockedIncre...	ds
004061ba	ReleaseSemaphore	"ReleaseSemapho...	ds
004061ce	SetUnhandledExc...	"SetUnhandledExc...	ds
004061ee	Sleep	"Sleep"	ds

You can also check the function graphs


Function Graph [CodeBrowser(2): Demo:/crackme0x01.exe]
File
Edit
Analysis
Navigation
Search
Select
Tools
Help
Function Graph - _main - 4 vertices (crackme0x01.exe)

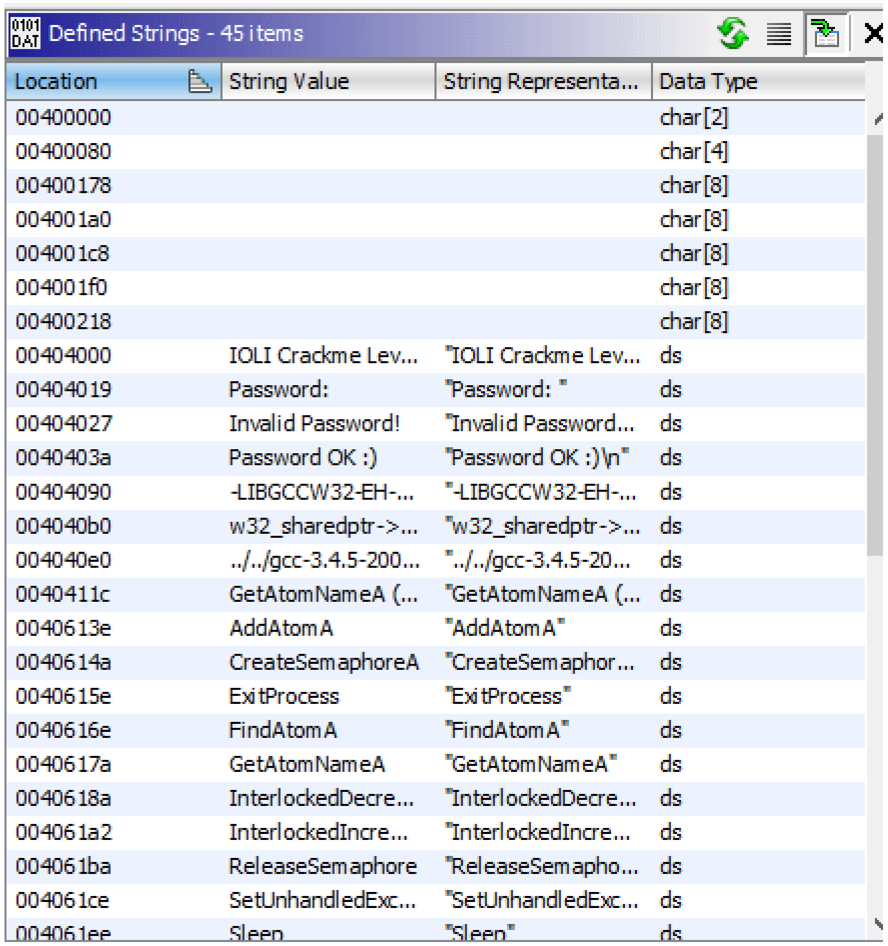
```
int __cdecl _main(int _Argc, char * * _Argv, char * * _Env
        int            EAX:4          <RETURN>
        int            Stack[0x4]:4   _Argc
        char * *       Stack[0x8]:4   _Argv
        char * *       Stack[0xc]:4   _Env
        undefined4     Stack[-0x8]:4  local_8
        undefined4     Stack[-0xc]:4  local_c
        undefined4     Stack[-0x1c]:4 local_1c
        undefined4     Stack[-0x20]:4 local_20
              .text
              _main
..1310 PUSH EBP
..1311 MOV  EBP,ESP
..1313 SUB  ESP,0x18
..1316 AND  ESP,0xfffffff0
..1319 MOV  EAX,0x0
..131e ADD  EAX,0xf
..1321 ADD  EAX,0xf
..1324 SHR  EAX,0x4
..1327 SHL  EAX,0x4
..132a MOV  dword ptr [EBP + local_c],...
..132d MOV  EAX,dword ptr [EBP + local...
..1330 CALL __alloca
..1335 CALL ___main
..133a MOV  dword ptr [ESP]=>local_20,...
..1341 CALL _printf
..1346 MOV  dword ptr [ESP]=>local_20,...
..134d CALL _printf
..1352 LEA  EAX=>local_8,[EBP + -0x4]
..1355 MOV  dword ptr [ESP + local_1c ]...
..1359 MOV  dword ptr [ESP]=>local_20,...
..1360 CALL _scanf
..1365 CMP  dword ptr [EBP + local_8],...
..136c JZ   LAB_0040137c
```

```
0040136e
..136e MOV  dword ptr [ESP]=>local_20,...
..1375 CALL _printf
..137a JMP  LAB_00401388
```

```
0040137c - LAB_0040137c
            LAB_0040137c
..137c MOV  dword ptr [ESP]=>local_20,...
..1383 CALL _printf
```

```
00401388 - LA...
            LAB_00401388
..1388 MOV  EAX,0x0
..138d LEAVE
..138e RET
```

To solve the challenge let’s first start with extracting the binary strings



Defined Strings - 45 items

Location	String Value	String Representa...	Data Type
00400000			char[2]
00400080			char[4]
00400178			char[8]
004001a0			char[8]
004001c8			char[8]
004001f0			char[8]
00400218			char[8]
00404000	IOLI Crackme Lev...	"IOLI Crackme Lev...	ds
00404019	Password:	"Password: "	ds
00404027	Invalid Password!	"Invalid Password...	ds
0040403a	Password OK :)	"Password OK :)\n"	ds
00404090	-LIBGCCW32-EH-...	"-LIBGCCW32-EH-...	ds
004040b0	w32_sharedptr->...	"w32_sharedptr->...	ds
004040e0	../../gcc-3.4.5-200...	"../../gcc-3.4.5-20...	ds
0040411c	GetAtomNameA (...	"GetAtomNameA (...	ds
0040613e	AddAtomA	"AddAtomA"	ds
0040614a	CreateSemaphoreA	"CreateSemaphor...	ds
0040615e	ExitProcess	"ExitProcess"	ds
0040616e	FindAtomA	"FindAtomA"	ds
0040617a	GetAtomNameA	"GetAtomNameA"	ds
0040618a	InterlockedDecre...	"InterlockedDecre...	ds
004061a2	InterlockedIncre...	"InterlockedIncre...	ds
004061ba	ReleaseSemaphore	"ReleaseSemapho...	ds
004061ce	SetUnhandledExc...	"SetUnhandledExc...	ds
004061ee	Sleep	"Sleep"	ds

As you can notice we get all the strings of the file. One of them is “Password OK :)”

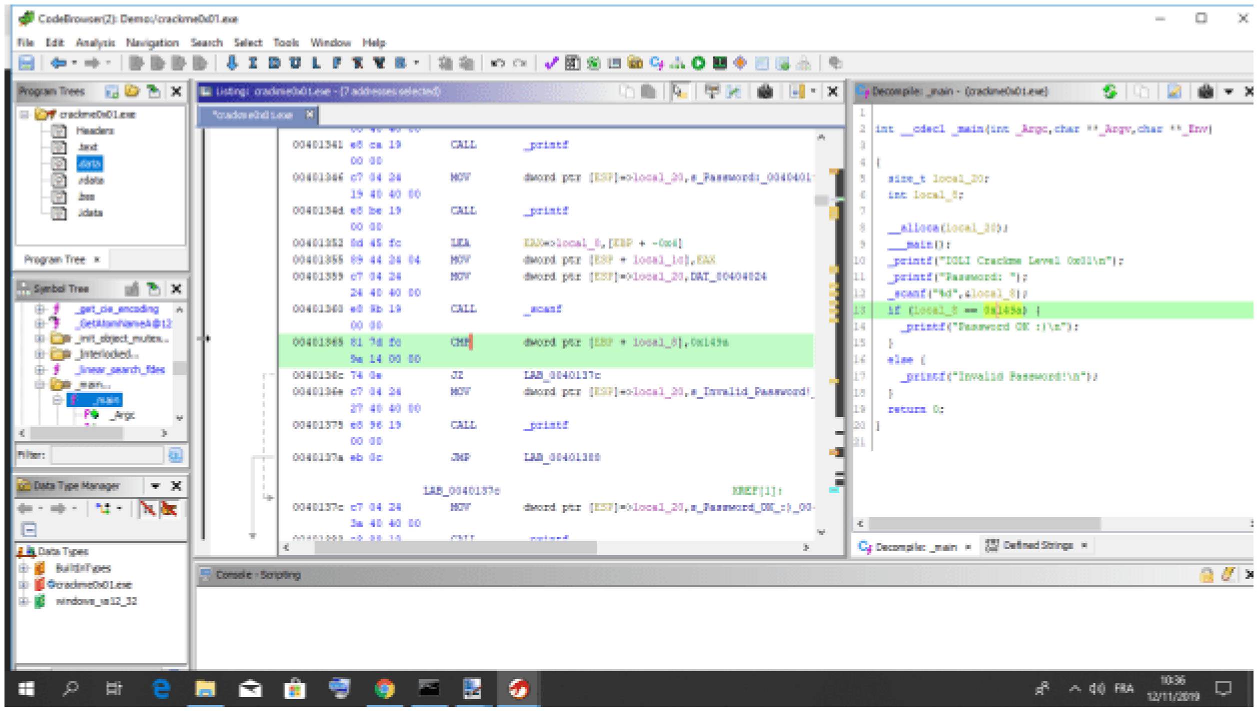
Ghidra is powerful. It gives you the ability to decompile the file. As you can see from the screenshot it is giving us a readable code.

If you check the code carefully you will notice this line of code

```
If (local_8 == 0x149a)

    _Printf ( “Password OK :) /n ”)
```

At the other side of the window you will see the CMP instruction. With a small Google search you will find that

*“CMP is generally used in conditional execution. This **instruction** basically subtracts one operand from the other for comparing whether the operands are equal or not. It does not disturb the destination or source operands. It is used along with the conditional jump **instruction** for decision making. ”*



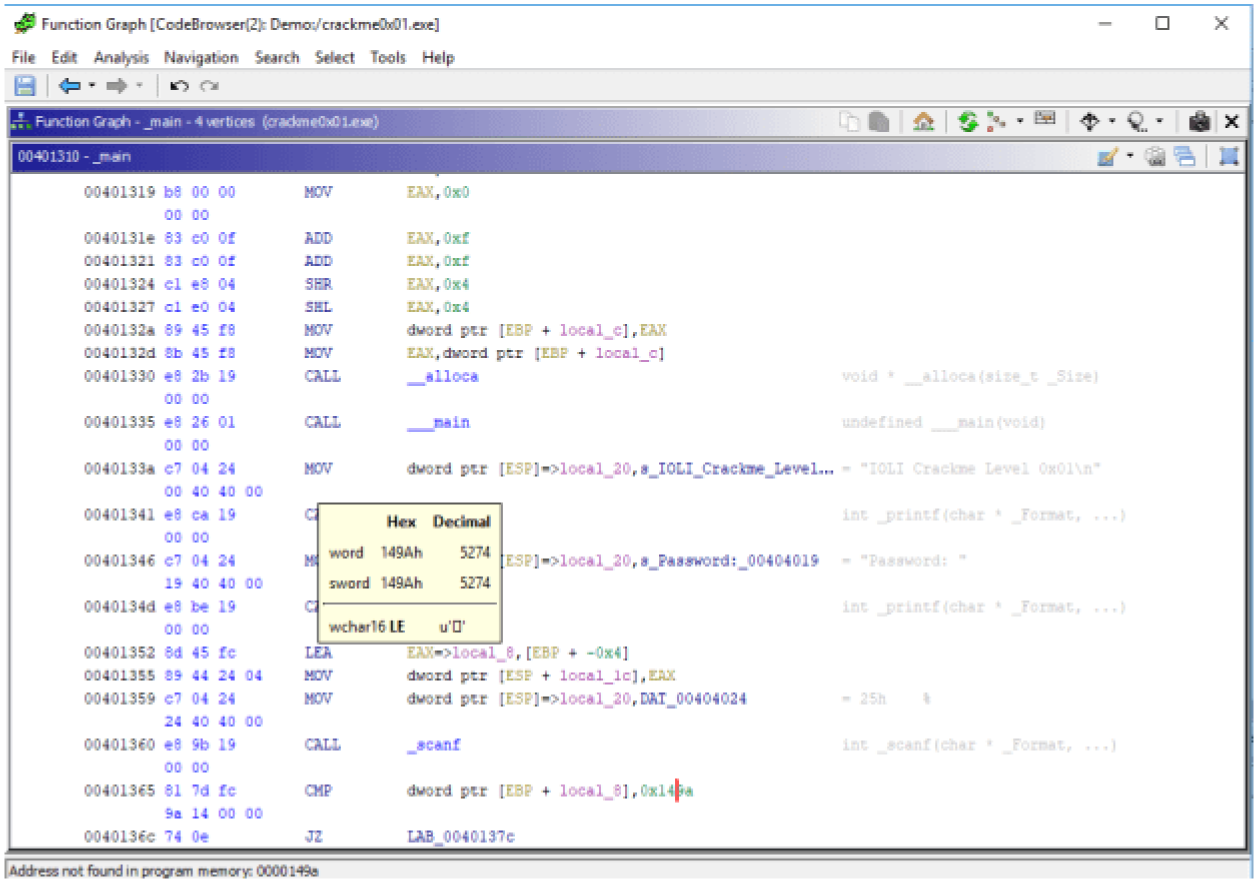
CodeBrowser(2): Demo:/crackme0x01.exe
File Edit Analysis Navigation Search Select Tools Window Help
Program Trees
Listing: crackme0x01.exe - (7 addresses selected)
Decompile: _main - (crackme0x01.exe)
Symbol Tree
Data Type Manager
Console - Scripting

```
00401341 e8 ca 19        CALL       _printf
         00 00
00401346 c7 04 24        MOV        dword ptr [ESP]=>local_20,s_Password:_0040401
         19 40 40 00
0040134d e8 be 19        CALL       _printf
         00 00
00401352 8d 45 fc        LEA        EAX=>local_8,[EBP + -0x4]
00401355 89 44 24 04     MOV        dword ptr [ESP + local_1c],EAX
00401359 c7 04 24        MOV        dword ptr [ESP]=>local_20,DAT_00404024
         24 40 40 00
00401360 e8 8b 19        CALL       _scanf
         00 00
00401365 81 7d fc        CMP        dword ptr [EBP + local_8],0x149a
         9a 14 00 00
0040136c 74 0e           JZ         LAB_0040137c
0040136e c7 04 24        MOV        dword ptr [ESP]=>local_20,s_Invalid_Password!_
         27 40 40 00
00401375 e8 96 19        CALL       _printf
         00 00
0040137a eb 0c           JMP        LAB_00401388

                 LAB_0040137c                               XREF[1]:
0040137c c7 04 24        MOV        dword ptr [ESP]=>local_20,s_Password_OK_:)_00
         3a 40 40 00
```

```
int __cdecl _main(int _Argc,char **_Argv,char **_Env)

{
  size_t local_20;
  int local_8;

  __alloca(local_20);
  ___main();
  _printf("IOLI Crackme Level 0x01\n");
  _printf("Password: ");
  _scanf("%d",&local_8);
  if (local_8 == 0x149a) {
    _printf("Password OK :)\n");
  }
  else {
    _printf("Invalid Password!\n");
  }
  return 0;
}
```

Then if our analysis is correct then the valid password will be a conversion of “0x149a”

To check its value double click on it and you will get this.


Function Graph [CodeBrowser(2): Demo:/crackme0x01.exe]
File Edit Analysis Navigation Search Select Tools Help
Function Graph - _main - 4 vertices (crackme0x01.exe)
00401310 - _main

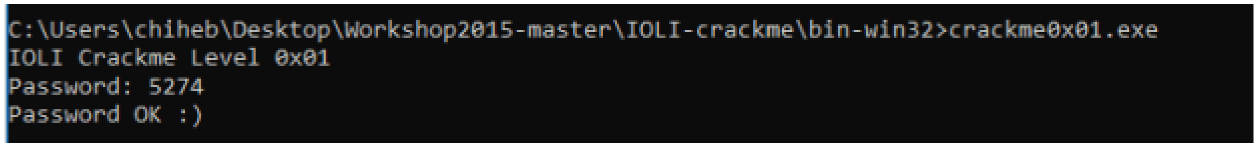
```
00401319 b8 00 00       MOV        EAX,0x0
         00 00
0040131e 83 c0 0f       ADD        EAX,0xf
00401321 83 c0 0f       ADD        EAX,0xf
00401324 c1 e8 04       SHR        EAX,0x4
00401327 c1 e0 04       SHL        EAX,0x4
0040132a 89 45 f8       MOV        dword ptr [EBP + local_c],EAX
0040132d 8b 45 f8       MOV        EAX,dword ptr [EBP + local_c]
00401330 e8 2b 19       CALL       __alloca                                  void * __alloca(size_t _Size)
         00 00
00401335 e8 26 01       CALL       ___main                                   undefined ___main(void)
         00 00
0040133a c7 04 24       MOV        dword ptr [ESP]=>local_20,s_IOLI_Crackme_Level... = "IOLI Crackme Level 0x01\n"
         00 40 40 00
00401341 e8 ca 19       C                                                    int _printf(char * _Format, ...)
         00 00
00401346 c7 04 24       M          [ESP]=>local_20,s_Password:_00404019      = "Password: "
         19 40 40 00
0040134d e8 be 19       C                                                    int _printf(char * _Format, ...)
         00 00
00401352 8d 45 fc       LEA        EAX=>local_8,[EBP + -0x4]
00401355 89 44 24 04    MOV        dword ptr [ESP + local_1c],EAX
00401359 c7 04 24       MOV        dword ptr [ESP]=>local_20,DAT_00404024    = 25h    %
         24 40 40 00
00401360 e8 9b 19       CALL       _scanf                                    int _scanf(char * _Format, ...)
         00 00
00401365 81 7d fc       CMP        dword ptr [EBP + local_8],0x149a
         9a 14 00 00
0040136c 74 0e          JZ         LAB_0040137c
```

	Hex	Decimal
word	149Ah	5274
sword	149Ah	5274
wchar16 LE	u'□'	

Address not found in program memory: 0000149a

The decimal value is “5274”. So let’s try it:

Go back to your terminal and run the binary and this time type 5274:



```
C:\Users\chiheb\Desktop\Workshop2015-master\IOLI-crackme\bin-win32>crackme0x01.exe
IOLI Crackme Level 0x01
Password: 5274
Password OK :)
```

Congratulations, you solved your first **crackme** challenge.

This article will be updated with more interesting sections in the next few hours like Malware Analysis with Ghidra

Further resources

- https://ghidra-sre.org/CheatSheet.html

References

- https://www.tutorialspoint.com/assembly_programming/assembly_conditions.htm

Summary

This article was a good opportunity to learn the fundamentals of reverse engineering with an amazing tool called "Ghidra"