TD 1 : Using Ghidra and Protecting Passwords

M1 CSSE / CyberUs – Secure Programming

Camille Monière (course manager, A. Pr.)   Maria Mendez-Real (teacher, A. Pr.)

November 2024

Foreword

The goal of the exercises is to give you context and a playground, to either learn techniques or use notions presented during the lectures.

You may continue to work on the exercises outside, but have to focus on them in class.

They are not graded, but may prepare you to the graded labs, and to the final exam.

Ghidra Summed Up

Ghidra is a retro-engineering tool. For short, it analyzes a binary executable file, and try to coerce the original source code.

It works well on C programs. In the context of secure programming, it allows to inspect the program as if we were attackers.

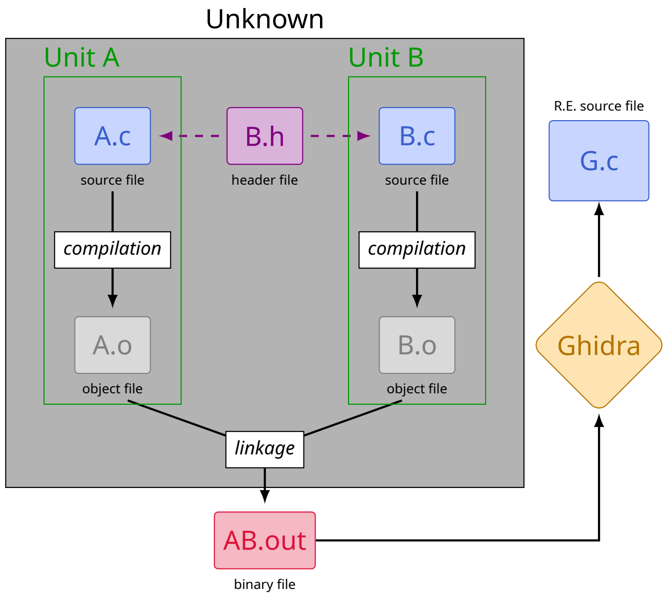


Figure 1: Ghidra location in a compilation flow

It is not a magical tool, and it sometimes needs help.

These exercises will give you some insights on the tool usage.

To use Ghidra, you need it installed on your machine, whether it is a University or personal one. Just grab an archive of the latest release, extract it somewhere, and run `<path-to>/ghidrarun` in the terminal.

It will open a page. If you already have a project, just use it. If you haven't, create a *non-shared* one. Then, import (*using* `I` *or the File menu*) the executable file you want to inspect.

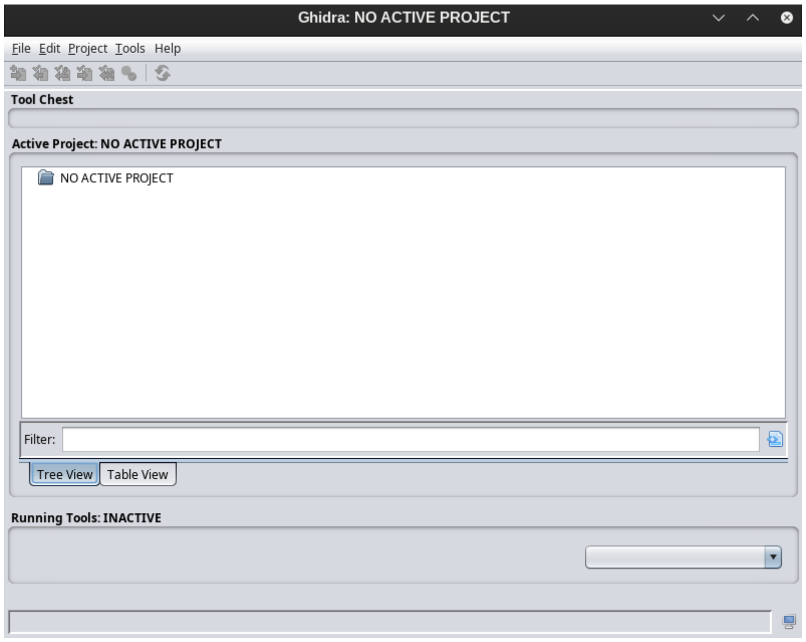
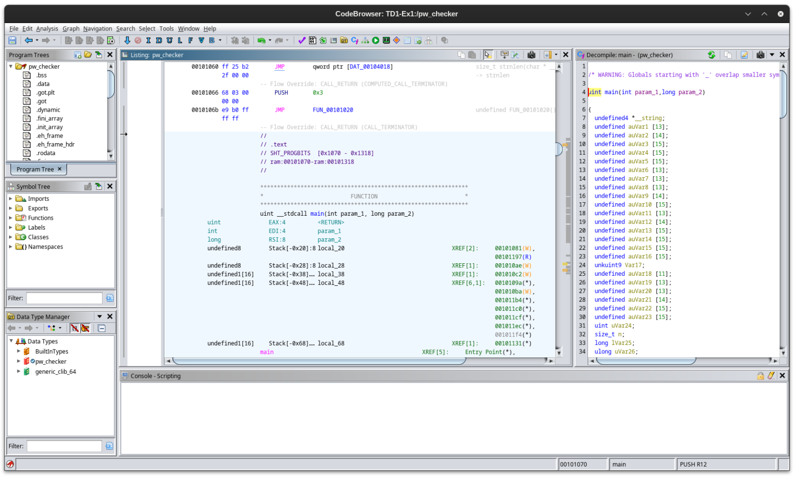

Ghidra: NO ACTIVE PROJECT
File Edit Project Tools Help
Tool Chest
Active Project: NO ACTIVE PROJECT
NO ACTIVE PROJECT
Filter:
Tree View
Table View
Running Tools: INACTIVE

Figure 2: Ghidra project page

You can now double-click on the imported file. When asked for analysis, accept. You should now have this kind of window :



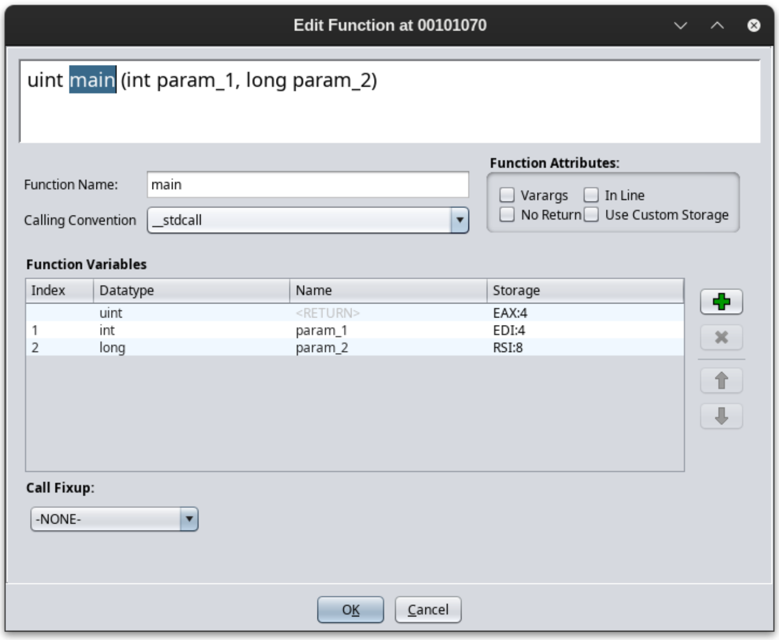
```
00101060 ff 25 b2       JMP        qword ptr [DAT_00104018]          size_t strnlen(char * _
         2f 00 00                                                    -> strnlen
                  -- Flow Override: CALL_RETURN (COMPUTED_CALL_TERMINATOR)
00101066 68 03 00       PUSH       0x3
         00 00
0010106b e9 b0 ff       JMP        FUN_00101020                      undefined FUN_00101020()
         ff ff
                  -- Flow Override: CALL_RETURN (CALL_TERMINATOR)
                  //
                  // .text
                  // SHT_PROGBITS  [0x1070 - 0x1318]
                  // ram:00101070-ram:00101318
                  //

                  *************************************************************
                  *                          FUNCTION                         *
                  *************************************************************
                  uint __stdcall main(int param_1, long param_2)
      uint            EAX:4          <RETURN>
      int             EDI:4          param_1
      long            RSI:8          param_2
      undefined8      Stack[-0x20]:8 local_20                  XREF[2]:     00101081(W),
                                                                            00101197(R)
      undefined8      Stack[-0x28]:8 local_28                  XREF[1]:     001010ae(W)
      undefined1[16]  Stack[-0x38]... local_38                 XREF[1]:     001010c2(W)
      undefined1[16]  Stack[-0x48]... local_48                 XREF[6,1]:   0010109a(*),
                                                                            001010ba(W),
                                                                            001011b4(*),
                                                                            001011c0(*),
                                                                            001011cf(*),
                                                                            001011ec(*),
                                                                            001011f4(*)
      undefined1[16]  Stack[-0x68]... local_68                 XREF[1]:     00101131(*)
                  main                                    XREF[5]:     Entry Point(*),
```

```
/* WARNING: Globals starting with '_' overlap smaller sym

uint main(int param_1,long param_2)

{
  undefined4 *__string;
  undefined auVar1 [13];
  undefined auVar2 [14];
  undefined auVar3 [15];
  undefined auVar4 [15];
  undefined auVar5 [15];
  undefined auVar6 [13];
  undefined auVar7 [13];
  undefined auVar8 [13];
  undefined auVar9 [14];
  undefined auVar10 [15];
  undefined auVar11 [13];
  undefined auVar12 [14];
  undefined auVar13 [15];
  undefined auVar14 [15];
  undefined auVar15 [15];
  undefined auVar16 [15];
  unkuint9 Var17;
  undefined auVar18 [11];
  undefined auVar19 [13];
  undefined auVar20 [13];
  undefined auVar21 [14];
  undefined auVar22 [15];
  undefined auVar23 [15];
  uint uVar24;
  size_t n;
  long lVar25;
  ulong uVar26;
```

```
00101070    main    PUSH R12
```

Figure 3: Ghidra code viewer

Here, you can begin the retro-engineering task. For instance, do you see the main function ? Right-click on it. It will make the next window appear. The signature of the main function is likely to be wrong. Here, the second argument is `long` instead of `char **`. Let us fix that. You can also set the calling convention to default, which is often better.



Edit Function at 00101070

```
uint main (int param_1, long param_2)
```

Function Name: main

Calling Convention: __stdcall

Function Attributes: Varargs [ ] In Line [ ] No Return [ ] Use Custom Storage [ ]

Function Variables

Index	Datatype	Name	Storage
	uint	<RETURN>	EAX:4
1	int	param_1	EDI:4
2	long	param_2	RSI:8

Call Fixup: -NONE-

Figure 4: Ghidra function editor

There is a lot more, but you have the essential. You can explore the code, function by function, variable by variable. Sometimes, you will need to reconstruct the code to understand it. Sometimes, the answer will be visible without a complete retro-fitting.

Instructions

In the first two exercise, you will deal with a different executable. You must find the problem(s) and explain how you have found and it (them). Then, you must propose a mitigation or better, a solution. In the third exercise, the original source codes and makefiles are given, so you can push your analysis (and fix propositions) further.

Exercise 1: Keeping secrets

The executable is `pw_checker_native`. It takes one argument of at most eight character, and returns `0` if it is the password. It returns `1` otherwise.

Q1. Prior knowledge

Without analyzing, can you already find one vulnerability ?

Q2. Cracking using analysis

Without using Ghidra, any decompiler / disassembler, or any inspection tool, propose a method to crack the program.

What are the limits of such method ?

Q3. Cracking using tools

Using Ghidra, provide a method to crack the program open.

Q4. Mitigations

In your opinion, is there a way to secure such program:

1. without modifying the source code ? If yes, give an example.
2. by modifying the program ? If yes, give an example.

Exercise 2: Handling secrets

The executable is `pw_checker`. It takes one argument, and returns `0` if it is the password. It returns `1` otherwise. The password can be from `8` to `40` characters. It must be accompanied by the `libpasswds.so` file, for it to work. As a constraint, you must consider this file as non-writable.

Q1. File analysis

First, let's see what we have got.

1. Use the command `ldd pw_checker`. For comparison, try it on `pw_checker_naive`. What does that tell you ?
2. What is `libpasswds.so` ?

Q2. Code analysis

Use Ghidra on the executable.

1. What is your first impression ?
2. Do you recognize functions / code snippets at the beginning and at the end of the code ?
3. What is the method used to compare the passwords ?
4. What is the name of the variable ?
5. What does that tell you on the content of `libpasswds.so` ?

Q3. Cracking using normal behavior

The mechanism behind the file `libpasswds.so` is not secured at all. To understand why, search for *LD_PRELOAD* on the internet.

1. What does *LD_PRELOAD* do on Linux ?
2. Knowing this and using the answers from the previous question, propose a method to bypass `pw_checker`.
3. Implement and test your method.

Q4. Mitigations

In your opinion, is there a way to secure such program:

1. without modifying the source code ? If yes, give several examples.
2. by modifying the program ? If yes, give several examples.

Going forward

Ask your supervisor. If you have finished and understood previous exercises, you will be granted access to the next level.