Reverse Engineering

Premise

- Students are given a C file to reverse engineer
  - NIST: T0411

Recommended Tools

- Ltrace
- Ghidra

Questions

- What is the *MD5 sum* of the "AdminLogin" file
- What is the correct *password* for the admin login
- What message is displayed after a successful login

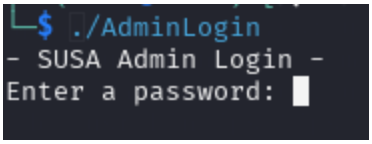
FILE

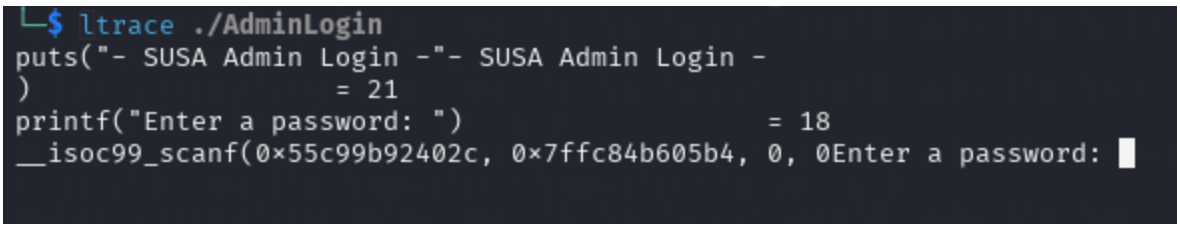
- AdminLogin

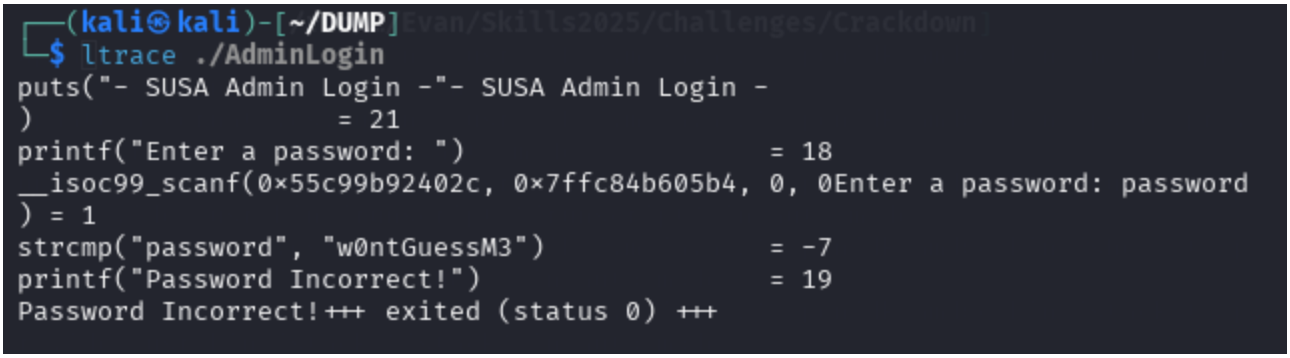
Answers

- What is the *MD5 sum* of the "AdminLogin" file
  - 1574c3a14e359e4bd8498ffe2c717152
- What is the correct *password* for the admin login
  - w0ntGuessM3
- What message is displayed after a successful login
  - :> Welcome Administrator

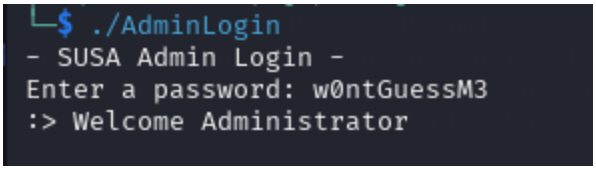
Walkthrough

1. There are two approaches to this challenge, using two different utilities.
2. Starting with ***ltrace*** as its simpler
    1. The *AdminLogin* can be run in the terminal with
        1. *./AdminLogin*
        2. 

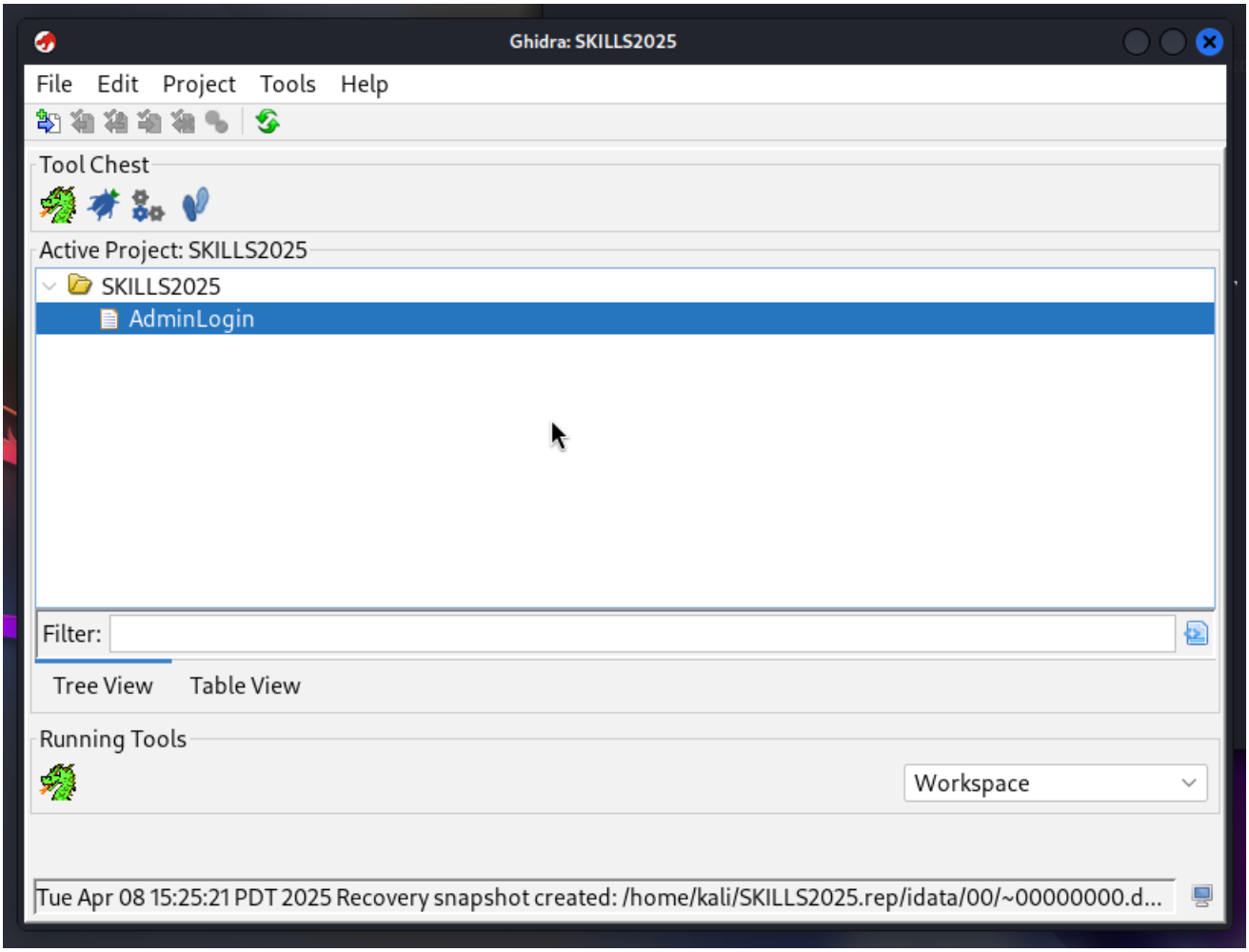
    2. *ltrace* is a program that looks through the stack as the program is running and returns memory addresses, calls, and values that are being accessed at run time
        1. Run the program with *ltrace*
        

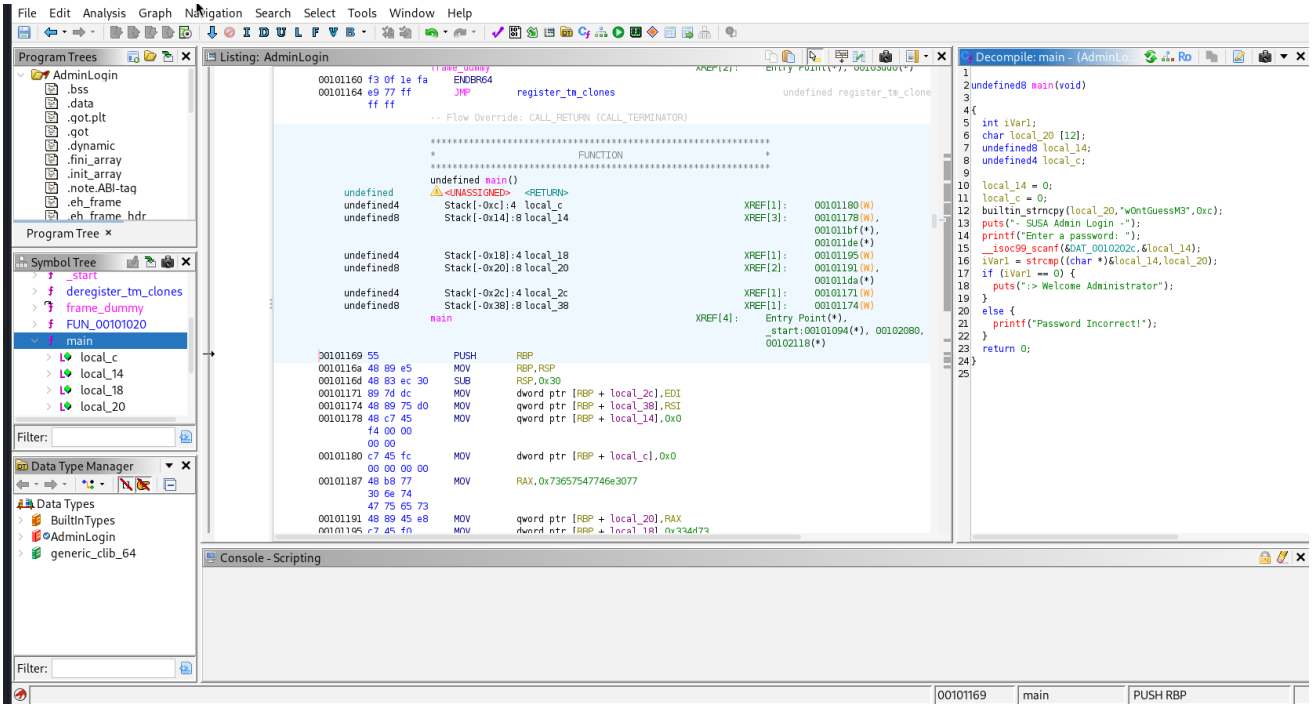
        2. This shows the base C code calls, printing the password prompt with *"printf()"* and then getting the users input with *"_isoc99_scanf()"* which is just a call to *"scanf()"*
    3. In this example we can assume that the users input will be taken in, and then compared to the password, so the password can be set to any value this execution
    

    4. *ltrace* will show that the program is comparing the users input to a value that is stored in the program
        1. This line which uses *"strcmp()"* which stands for string compare, its checking the users input against the password string that is saved in the program
        

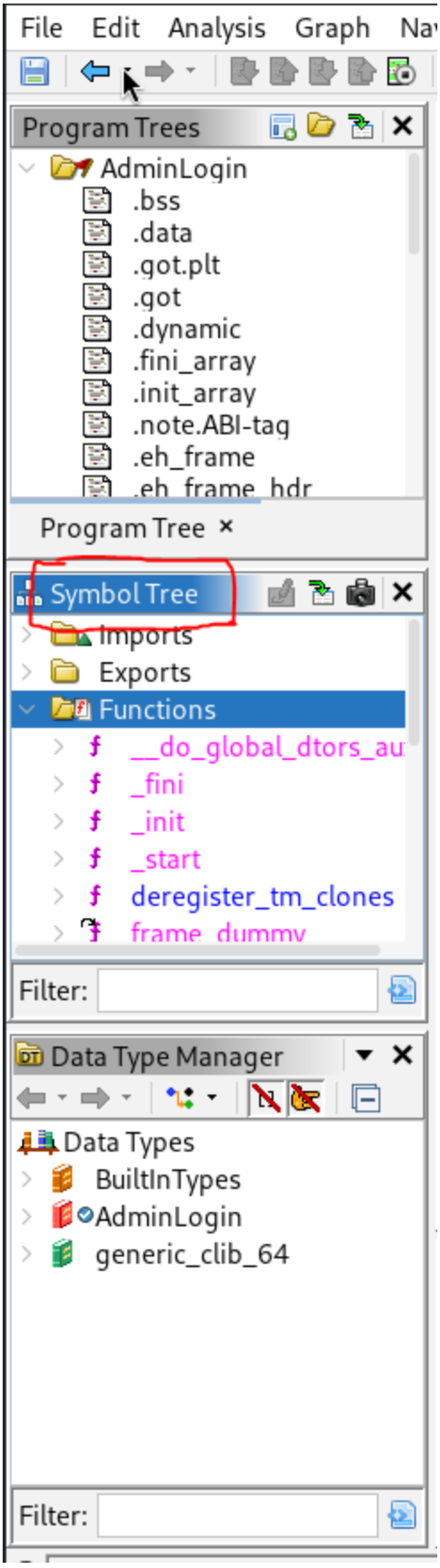
        2. From this snippet we can see that "*password*" is compared against "*w0ntGuessM3*"
            1. The string compare operator is vulnerable in this way, as it needs to load values into memory in order to compare them. This load means that the strings are copied to another memory location instead of just referencing their original memory address, allowing ltrace to view the password
    5. Running the program again and entering "*w0ntGuessM3*" as the password will yield the correct result
        1. 

3. Not for ***Ghidra***

1. Open up a new terminal and type *"ghidra"* and press enter
2. Load up a new project and import the '*AdminLogin*' file into the ghidra file explorer
    1. 

    2. Double click "*AdminLogin*" file to open it in the decompiler
3. Ghidra will analyze the program automatically as a Linux ELF binary, nothing needs to be changed here.
4. When the project is opened Ghidra will go line by line in the assembly and translate it to a human readable format (In this case its C)
    1. 

5. Search under the "Functions" tab in the "Symbol Tree"



1.

2. Then look for the "main" function label

6. Ghidra will present you with the bare assembly in the middle of the screen

```
                    -- Flow Override: CALL_RETURN (CALL_TERMINATOR)

                    *************************************************************
                    *                          FUNCTION                         *
                    *************************************************************
                    undefined main()
          undefined       <UNASSIGNED>   <RETURN>
          undefined4        Stack[-0xc]:4  local_c                    XREF[1]:     00101180(W)
          undefined8        Stack[-0x14]:8 local_14                   XREF[3]:     00101178(W),
                                                                                   001011bf(*),
                                                                                   001011de(*)
          undefined4        Stack[-0x18]:4 local_18                   XREF[1]:     00101195(W)
          undefined8        Stack[-0x20]:8 local_20                   XREF[2]:     00101191(W),
                                                                                   001011da(*)
          undefined4        Stack[-0x2c]:4 local_2c                   XREF[1]:     00101171(W)
          undefined8        Stack[-0x38]:8 local_38                   XREF[1]:     00101174(W)
                    main                                        XREF[4]:     Entry Point(*),
                                                                             _start:00101094(*), 00102080,
                                                                             00102118(*)

     00101169 55            PUSH        RBP
```

1. This is the program as the computer sees it, assembly is the finaly layre of abstraction before binary which is true machine code

7. To the right of that will be Ghidras decompiled version of that assembly

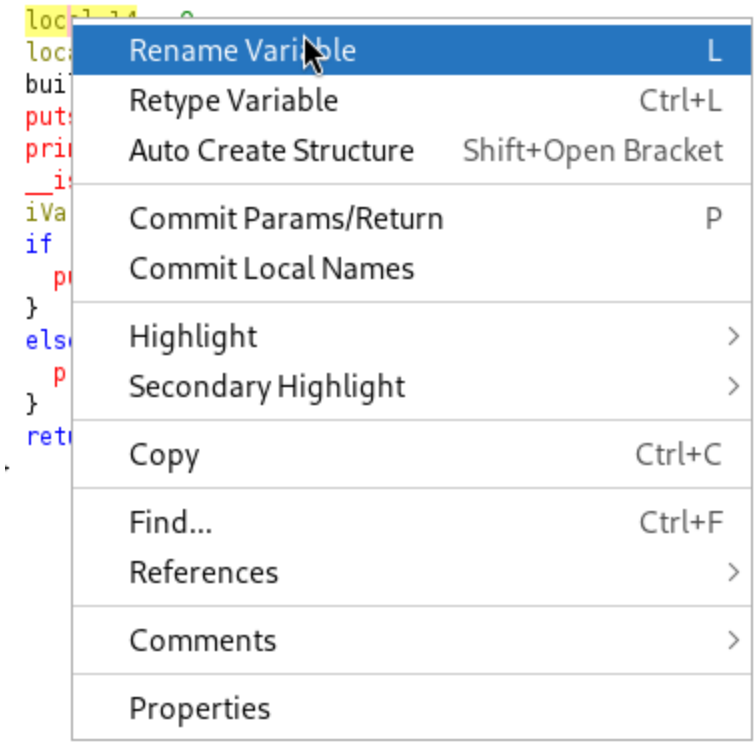
Decompile: main - (AdminLo...

```
1
2 undefined8 main(void)
3
4 {
5   int iVar1;
6   char local_20 [12];
7   undefined8 local_14;
8   undefined4 local_c;
9
10  local_14 = 0;
11  local_c = 0;
12  builtin_strncpy(local_20,"w0ntGuessM3",0xc);
13  puts("- SUSA Admin Login -");
14  printf("Enter a password: ");
15  __isoc99_scanf(&DAT_0010202c,&local_14);
16  iVar1 = strcmp((char *)&local_14,local_20);
17  if (iVar1 == 0) {
18    puts(":> Welcome Administrator");
19  }
20  else {
21    printf("Password Incorrect!");
22  }
23  return 0;
24 }
25
```

2.

3. This is the main program in a human readable format

8. Looking through the program can be slightly confusing but there are tools within Ghidra to help make it a little more readable
    1. Right clicking on any variable brings up a menu, renaming functions is probably the most



    useful feature for this use case
    2. When renaming variables its often very helpful to look at where functions are referenced, how they are used, and where they might end up being used. Adding context to variable names makes reverse engineering a much simpler task
9. After renaming some functions the code becomes a little more readable

```
undefined8 main(void)

{
  int Check;
  char UserInput [12];
  undefined8 String1;
  undefined4 Counter2;

  String1 = 0;
  Counter2 = 0;
  builtin_strncpy(UserInput,"w0ntGuessM3",0xc);
  puts("- SUSA Admin Login -");
  printf("Enter a password: ");
  __isoc99_scanf(&DAT_0010202c,&String1);
  Check = strcmp((char *)&String1,UserInput);
  if (Check == 0) {
    puts(":> Welcome Administrator");
  }
  else {
    printf("Password Incorrect!");
  }
  return 0;
}
```

    1.
    2. We can see the main purpose of this code is to use a string compare function, we can see the password stored in plain text on on line 12

10. Finally the password is revealed

```
11  Counter2 = 0;
12  builtin_strncpy(UserInput,"wOntGuessM3",0xc);
13  puts("- SUSA Admin Login -");
```