# Assignment Objectives

* See if any overflows
* See if nasty strings break
* Find Usernames
* Find Passwords
* Create modified binary to automatically get access

Issues

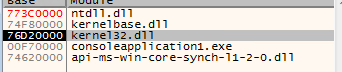
* Heap
* Naming function calls
* Return values/storage function calls
* Restarting exe/address changes
* Python script with pipes to call your program

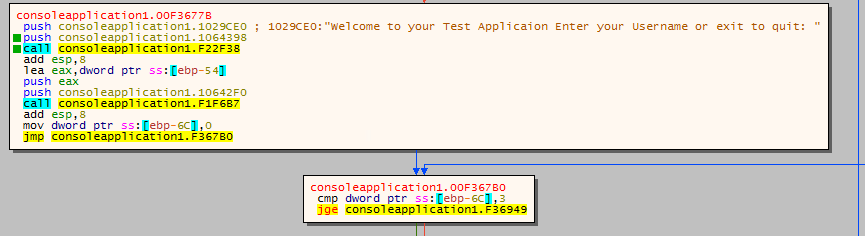
What I’ve do so far

* Execution map(roughly know functionality)
* Users names and passwords
* Found the places you are comparing user input to see if username or password is correct
* Played around with decompiler: looks like you a big while loop, allocate space for usernames and passwords, and have a couple if statements to find username and password, then tell them they are logged in once correct.
* More familiar with x64dbg and assembly in general

Questions:

* How can we easily view the Heap?
  + You can search in the symbol table for “malloc” and then set a breakpoint on it, but there are tons of calls to malloc when printing out things and using other standard library functions, so that doesn’t help that much.
* How can we view Global/Static variables?
  + Looks like most number variables are in .data. just translate from hex to decimal.
  + Looks like read-only data, such as literal strings, constants, and debug directory information are in .rdata
* How can we view all custom made function calls?
  + The .pdb file would tell you all of the useful debugging info, but we don’t have that, we just have the executable. This is the Microsoft file type for debugging windows apps.
  + Function calls with lower addresses are user defined ones though:



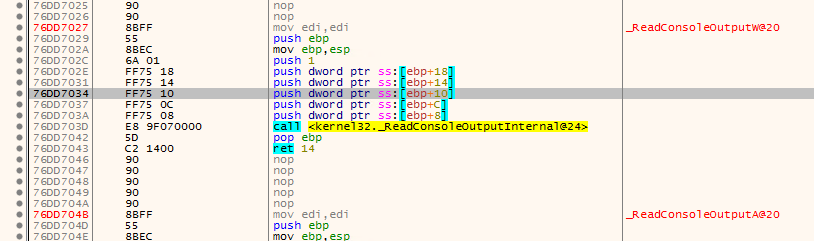
See how the .dll’s are 7 addresses where as the custom.cpp application is at address x00F22F38… 

Well if we look at the memory map, these functions are in the .text section – which contains machine code of all functions we would have written in C sourcefile.



So did Professor Randall just write his own GetConsoleInput functions to screw with me? No.

* + For external libraries used in the application: you can right click when in the external module and chose “download symbols for this module”. This will then fillout the names of all of the functions in the “comment” section of the disassembler.



* + When the application prompts for input, we are in a x777~ memory address which means its an external library function. It always says “kernel32” by EIP
* How can we mark common c library functions so we know what “call \*function:asdf\*” means in the assembly code? I assume there are some string compare calls or something going on here.
* Where do function calls store their results?
  + Points and Dump:
  + Where is my input stored?
  + Where is it being compared to the correct value?
* Is there a better way to “restart” the program, redragging the menu is annoying, and not sure if the addresses change.
* How can I find out how much space was allocated for the username and password? Feeding too many characters to username breaks the application

# Walkthrough

## Gathering Information

## Fuzzing the Application

## Controlling Execution

## Patching

# Extra Information

Execution Breakpoint:

This is the most common and most used type of breakpoint. When you toggle a breakpoint on a specific address, this tell the debugger to stop when that address is reached in the execution. To use this, simply press the F2 when over an address you would like to break on.

Memory Breakpoint:

A memory breakpoint is used to pause an application when a specific area of memory is either accessed, written to, or executed. This is handy when you want to determine when or if a specific area of memory is used by the program. This is available in the right click menu of the memory map window and dump pane.

Hardware Breakpoint:

A hardware breakpoint is used to pause an application when a particular address is either written to, read, or executed. This is specifically useful to determine when a particular variable is set. This can be used for byte, word, and dword reads and writes. This feature is available in the right click menu of the hex dump.

The Call Stack window:

Gives you an extensive list of the functions and procedures (routines) which brought you to your current location. You can use this window to analyze these routines and learn about your application's execution routine.

Memory Map:

This will show us all of the sections or regions of RAM allocated to and by our application and its dependencies.