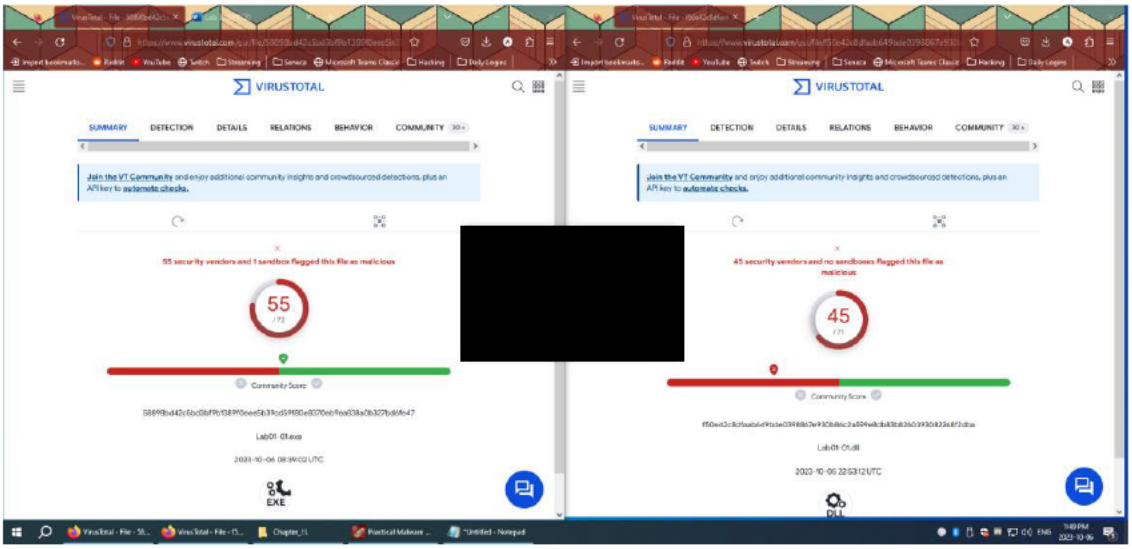
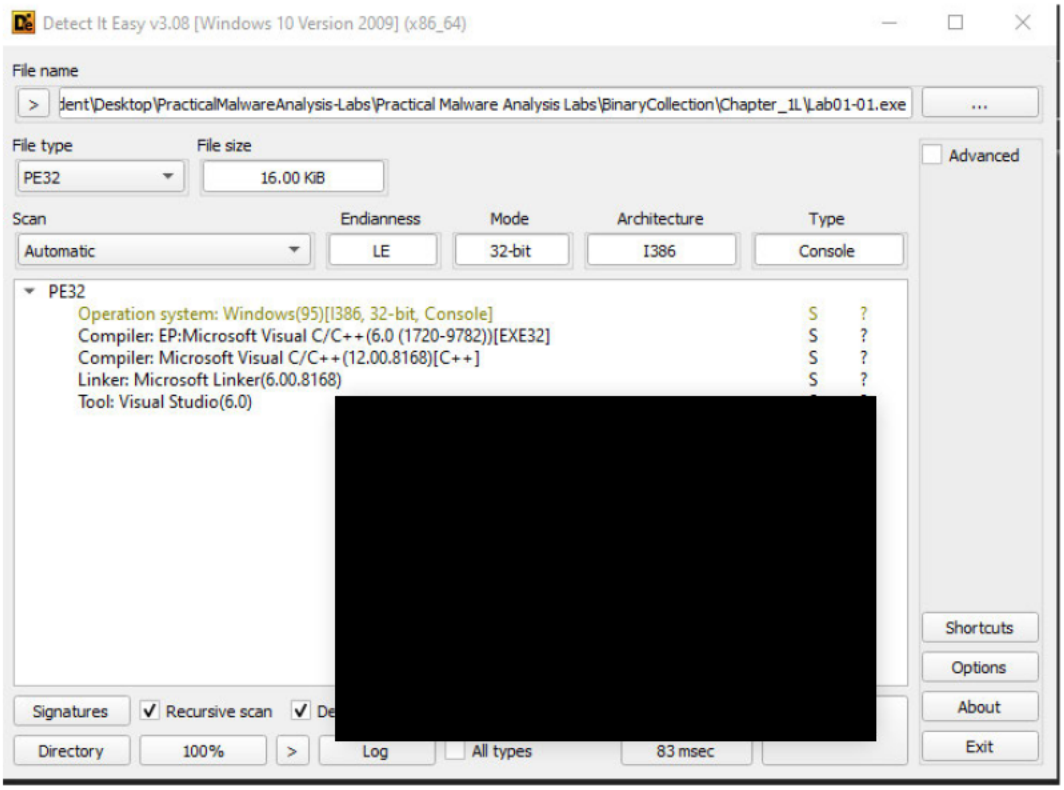
Task 1

Lab 1.1

1. Both files had numerous vendors flag the existing antivirus signatures.



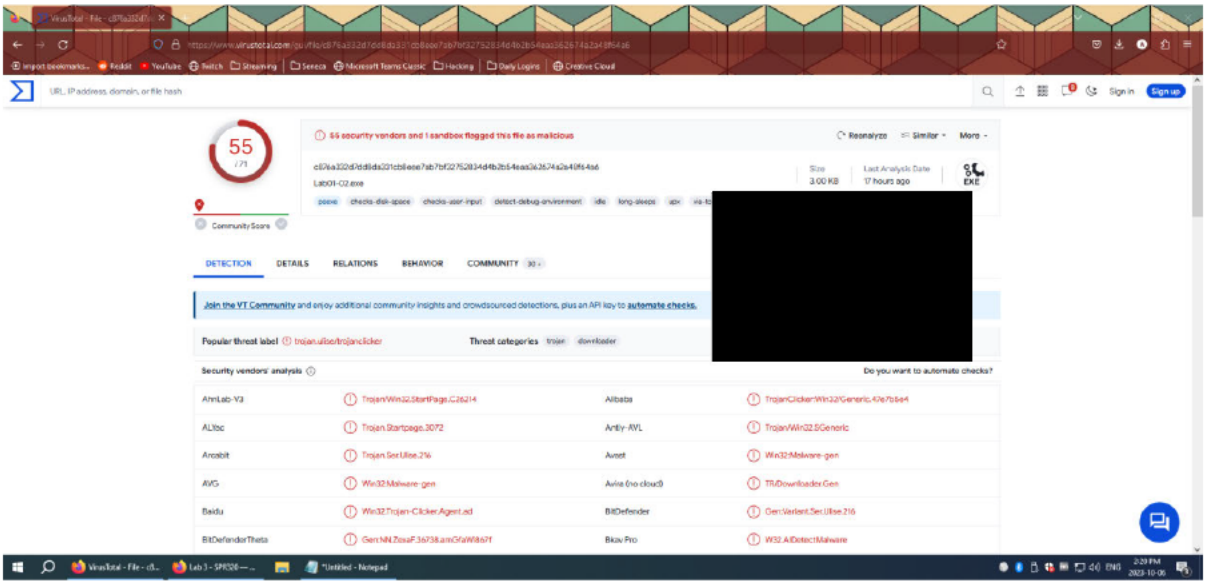
2. These files were compiled on December 19th, 2010 at 16:16:38UTC this was found in the details section on Total Virus
3. When analyzing the files with the Detect It Easy program we can see that there is no packer found.



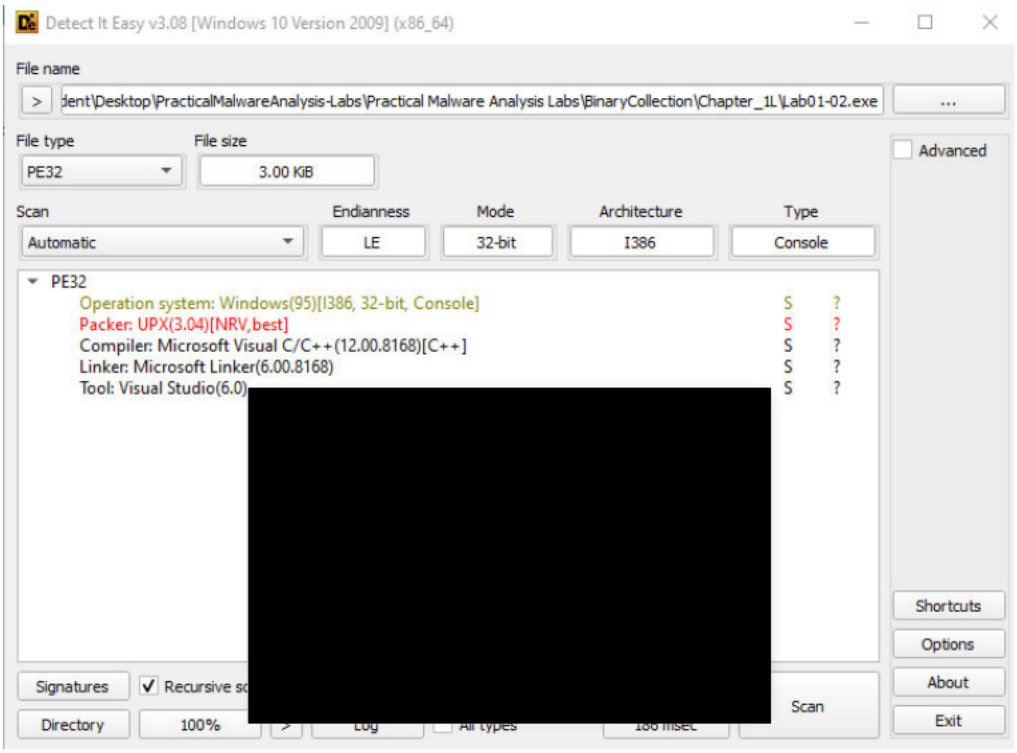
4. Based upon the imports found in .exe scan from VirusTotal, one can be the assumption that the program creates a File, maps to that file, then searches the target machine with the findfirstfileA and findnextfileA imports. The .dll file utilizes numerous MutexA calls (learned about through this resource) which creates a pointer within a security structure. Perhaps in conjunction the virus maps the files of target system.
5. There are a few checks we can make to see if a system is infected, the .exe uses multiple imports that create files so finding unfamiliar files can be a host-based indicator. Moreover, the use of WSACleanup and WSAStartup are indicators that something is utilizing dll services.
6. The .dll also uses closesocket so if a socket is meant to be opened but found close can be a indicator that the .dll was run which can be a network-based indicator.
7. In my guess, the virus creates a map of the user's file system, then uses tools like Sleep, shutdown to hide itself. It also uses some internet connection settings (like the inet_addr) whose purpose I cannot pin down, perhaps makes a record of the user's ipv4 information.

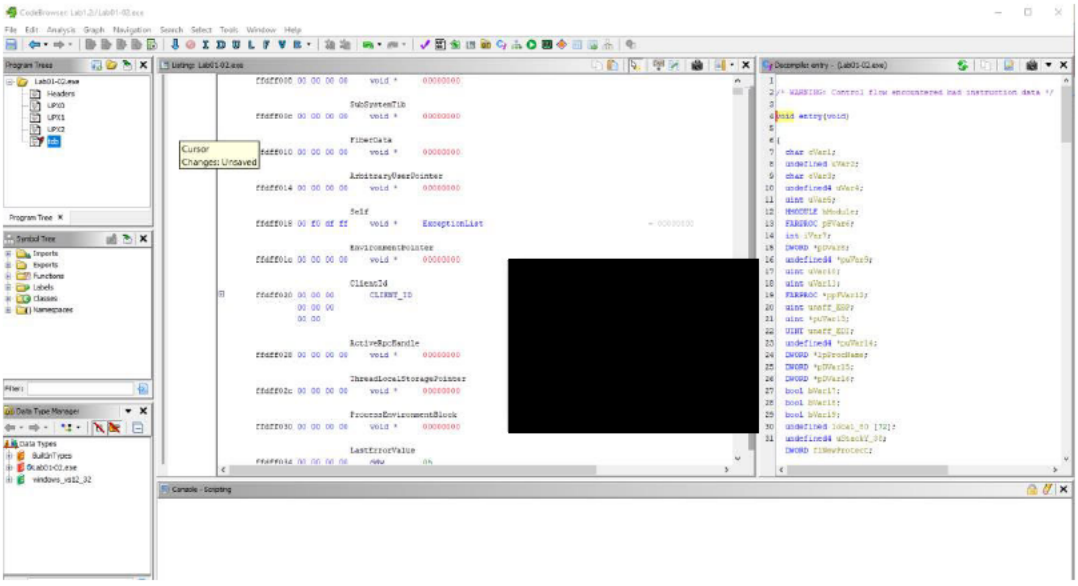
Lab 1.2

1. The virus matches numerous antivirus definitions being labelled as a trojan clicker as its "Popular Threat Label"



2. Through the Detect It Easy software, we can see this file is packed with the UFX packer. The file was unpacked with Ghidra

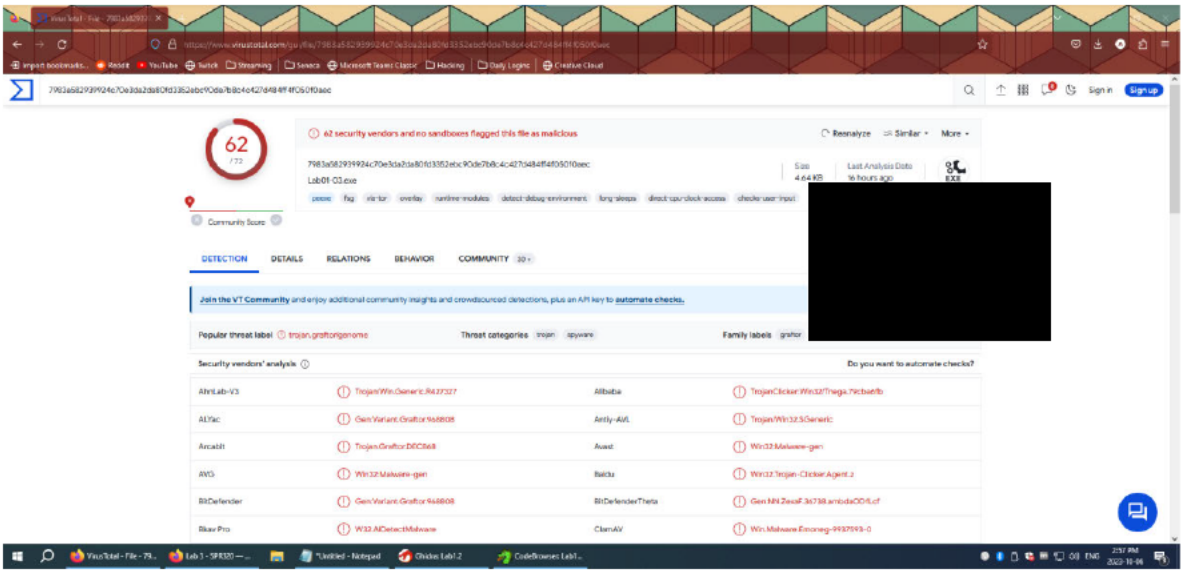




3. Looking at some of the variables that imported, the trojan creates a service in the ADVAPI32.dll then grabs an external service with the LoadLibraryA command, finds an external IP with the GetProcAddress, however, to how these imports work together I am unsure.
4. There are a few host-based indicators that could identify the IP, it makes a call with the GetProcAddress which would have to register through the router attached to your network. Perhaps a tool like WireShark could catch this, or a firewall could block this.

Lab 1.3

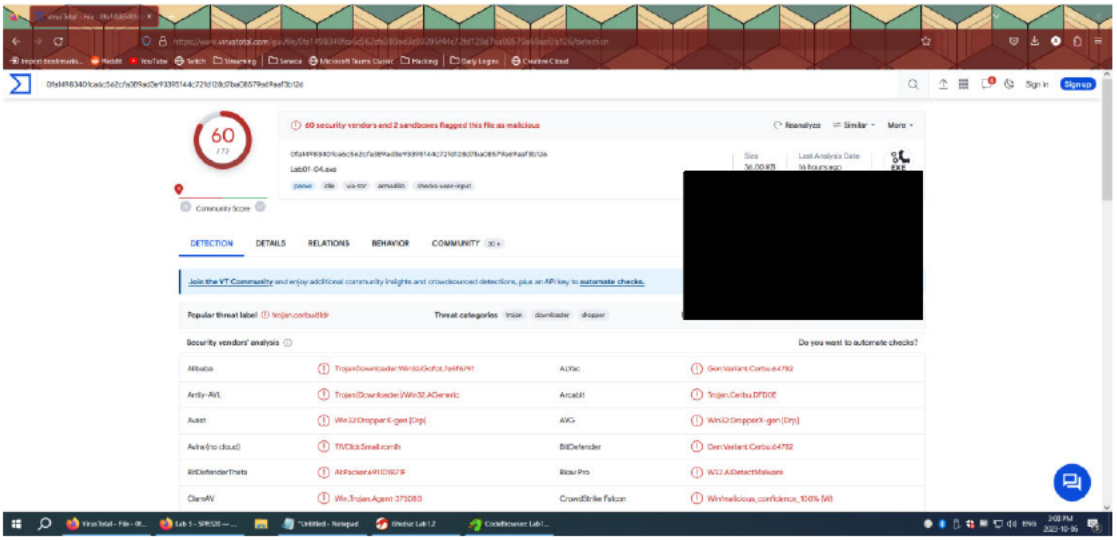
1. This file as well matches numerous definitions. With the tags of trojan spyware.



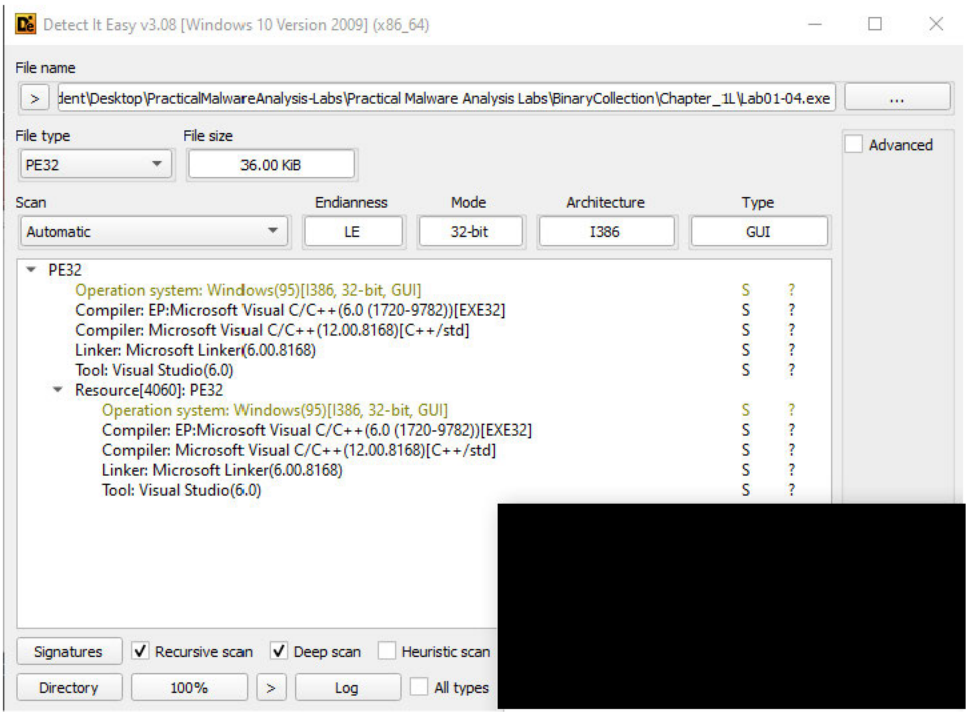
2. This file does not appear to be packed or obfuscated.
3. Due to the lack of imports present in this file, it seems that the purpose of this malware is to create a means of getting malicious software onto the end user's device. To support this theory, the use of GetProcAddress in addition to LoadLibraryA are used which find an IP address and possibly load the library from that.
4. As mentioned above the GetProcAddress making calls to an unknown IP could be a large indicator used to identify this malware. Perhaps a list of running services could also be employed.

Lab 1.4

1. The .exe does match numerous antivirus definitions as shown in the screen capture below.



2. By scanning the file with Detect It Easy, the virus is not packed or obfuscated in any way.

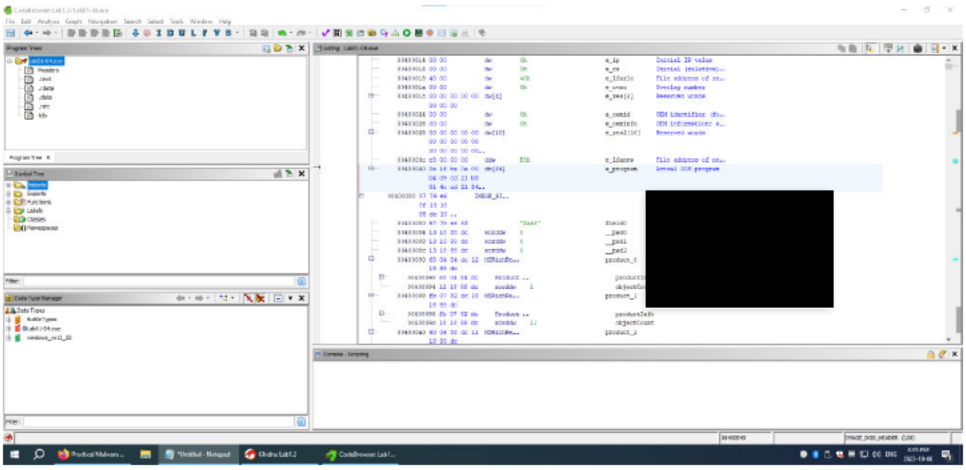


3. As per TotalVirus, the program was compiled on August 30th, 2019

Header

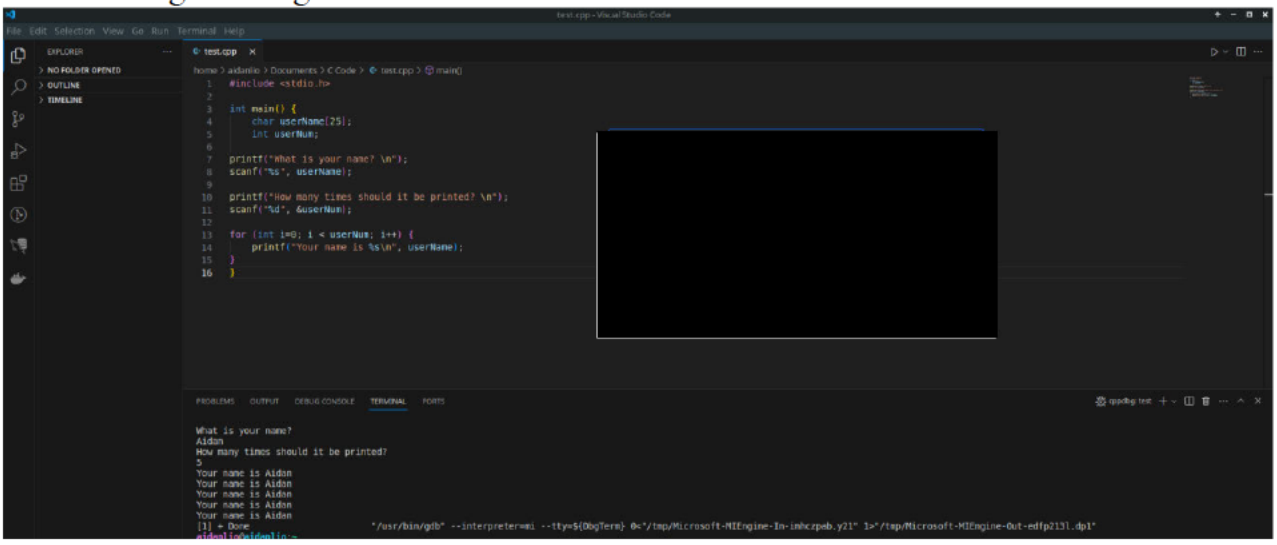
Target Machine	Intel 386 or later processors and compatible processors
Compilation Timestamp	2019-08-30 22:26:59 UTC
Entry Point	5583
Contained Sections	4

4. This .exe is definitely the most verbose of the 4 scanned, in the ADVAPI32.DLL it uses an access token to escalate prvilege (lookup + adjust + open?). The .exe creates a file and creates a remote thread (possibly to mask itself), there are other imports like GetTempPath and getWindowsDirectory that imply it creates a resource (possibly mapping a library or the processes on one's device) then moves the files (MoveFileA) and writes those files after using the SizeOfResource import.
5. Many of these imports are locally based, however, the .exe does make calls out with the LoadLibraryA, LoadResource, so it may be possible to catch the virus making calls onto your network.
6. In Gihdra, I noted a section stating Actual DOS program within the header of the file. When analyzing the bits, we see a translation into ASCII of "This Program Cannot be Run in DOS Mode". I believe, it is in this section of the header that the file is actually run, as highlighting this section of the ASCII text corresponds to the entire program within Ghidra.



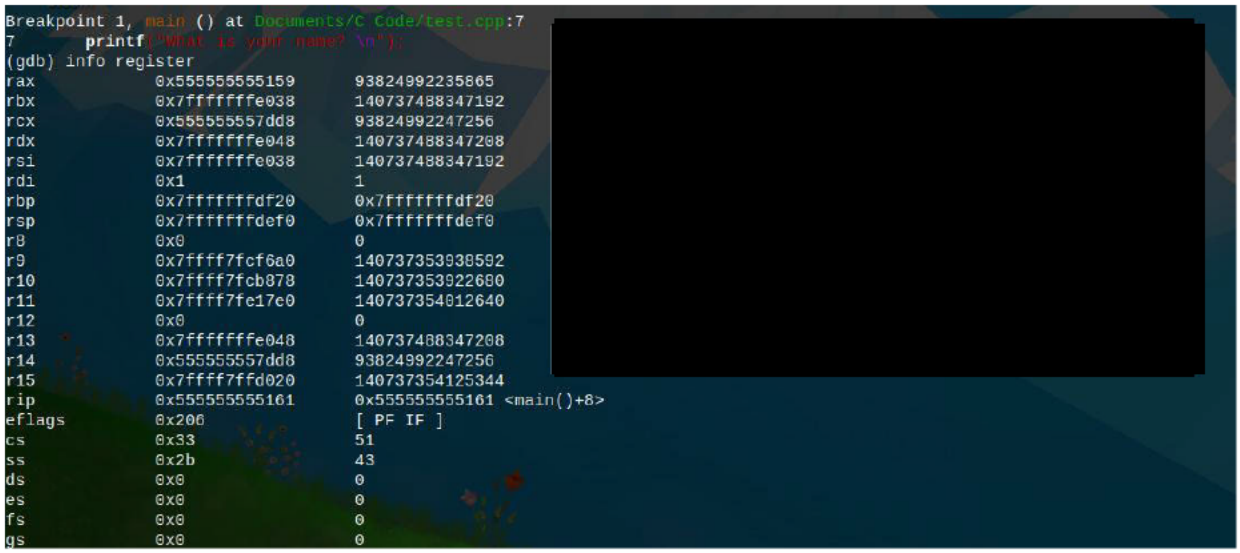
Task 2

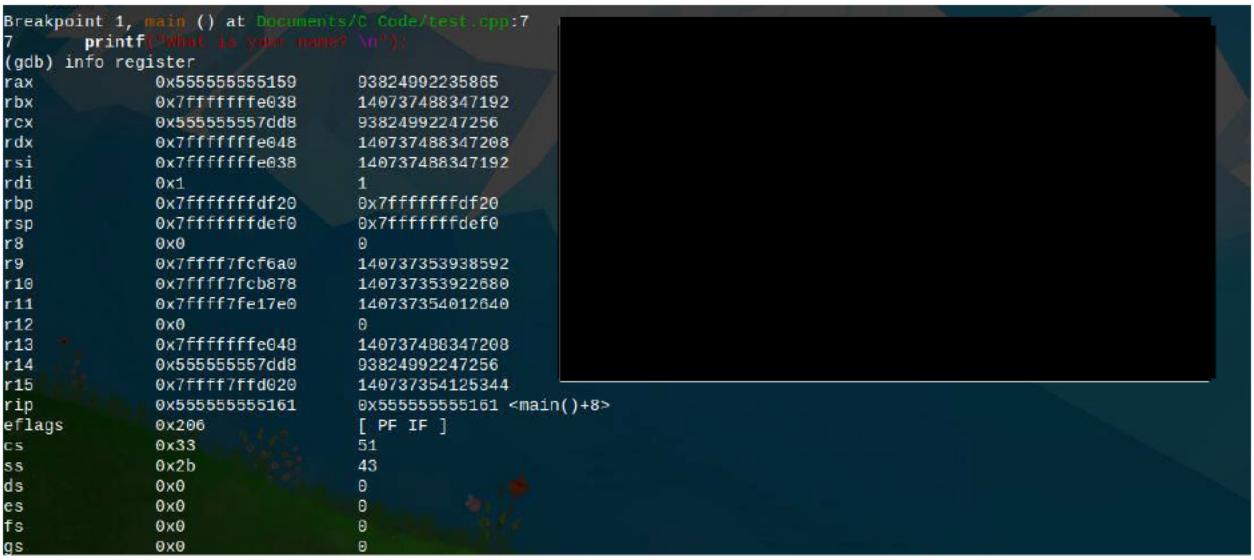
Basic C Programming



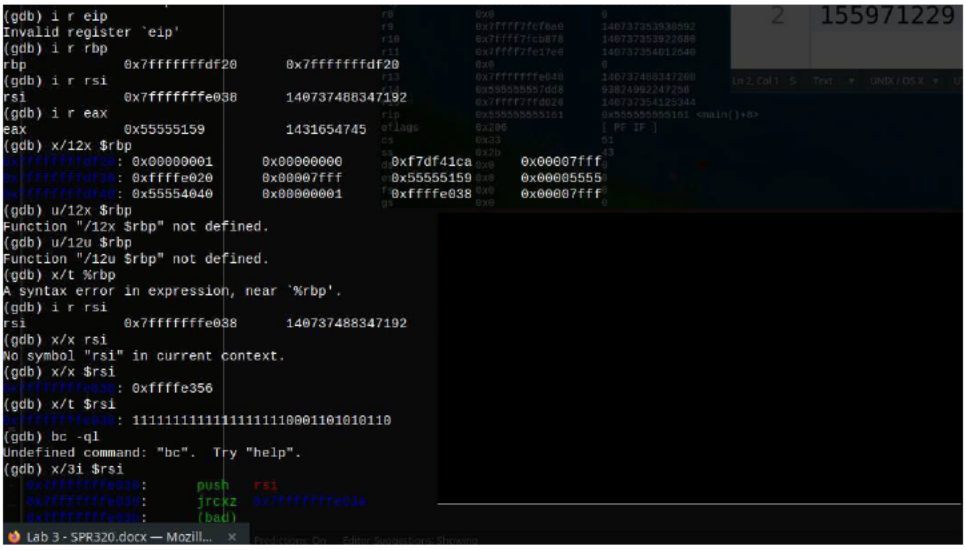
The above code uses a basic for loop to print the name of the user a specified number of times. Using a variable i as a basis for how many times the loop is run being incremented (i++ each time). Both the username and number of loops are inputted by the user and stored using the scanf command, storing them into variables with specific identifiers (%s for char and %d for an int).

Scanning the code following the steps utilized in 2X253 yielded some interesting learning regarding compilation as well as the steps required for even a rudimentary code as the one created above. After setting the disassembly-flavour I ran through the steps outlined on 25-27. My code did not feature an value for EIP (as shown below)

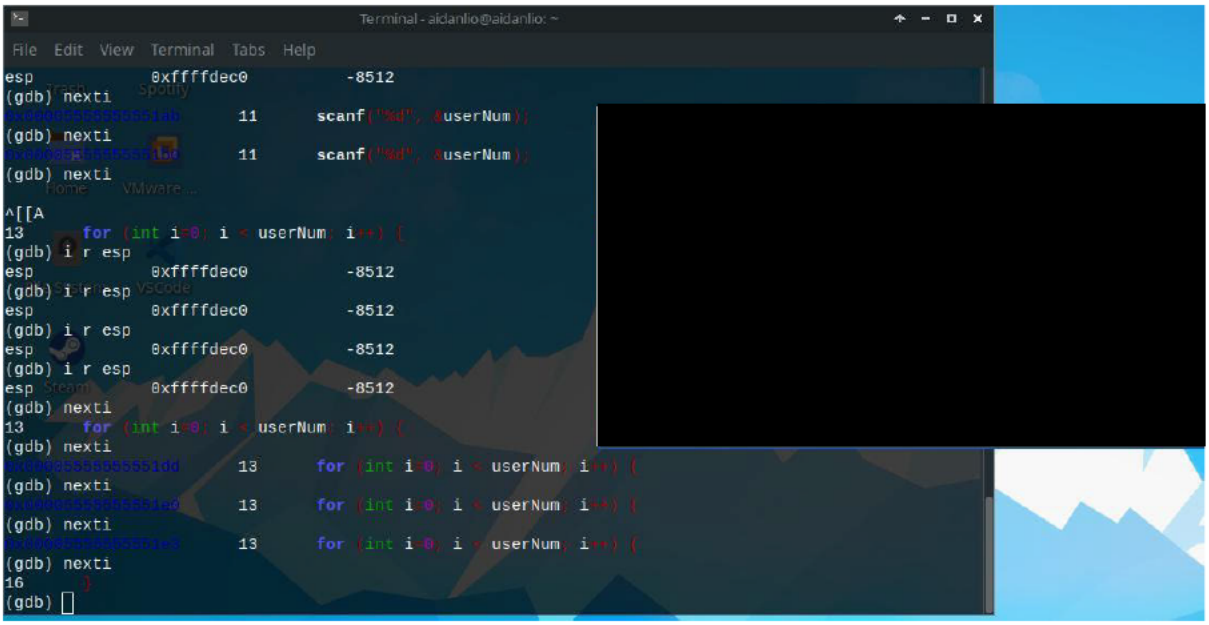




Although, this did present me with a multitude of information to look into. The below screenshot features me testing out various commands to garner some insight to how the gbd works.



To highlight an interesting facet is the determining of information stored using the printf() function, this is featured on page 35-36, however I was unable to utilize the *nexti* command in my loop as gdb would jump from line 13 to 16 (jumping over my loop) perhaps this is a biproduct of no-user input in the reviewing process. Something I am interested in is whether registers that are pointed to (link in the picture below) will have more to yield from if I am able to inject user inputs when going through the code in gdb.


Terminal - aidanlio@aidanlio: ~
File Edit View Terminal Tabs Help
esp 0xffffdec0 -8512
(gdb) nexti
11 scanf("%d", &userNum);
(gdb) nexti
11 scanf("%d", &userNum);
(gdb) nexti
^[[A
13 for (int i=0; i < userNum; i++) {
(gdb) i r esp
esp 0xffffdec0 -8512
(gdb) i r esp
esp 0xffffdec0 -8512
(gdb) i r esp
esp 0xffffdec0 -8512
(gdb) i r esp
esp 0xffffdec0 -8512
(gdb) nexti
13 for (int i=0; i < userNum; i++) {
(gdb) nexti
13 for (int i=0; i < userNum; i++) {
(gdb) nexti
13 for (int i=0; i < userNum; i++) {
(gdb) nexti
16 }
(gdb)