Lab: Cracking with Immunity

Environment

- Windows Virtual Machine
- Immunity Debugger
- Resource Hacker

Links for the required tools are available in the repository under the 'tools' directory.

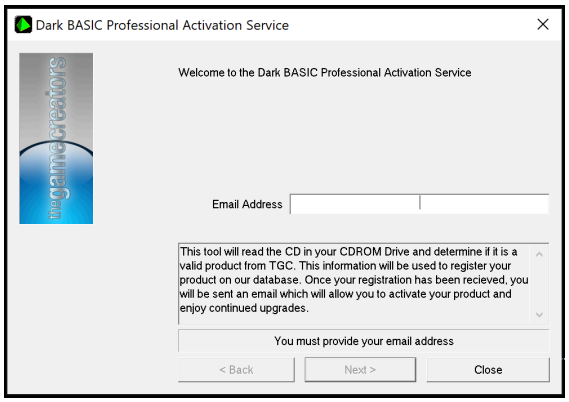
Target

- Dark Basic Pro

Provided in the 'target' directory.

Outline

1. Dark-Basic-Pro was a popular tool for creating Windows video games. It contains several different licensing schemes, one of which is a CD check to confirm that you have a valid copy of the install CD in the CDROM.



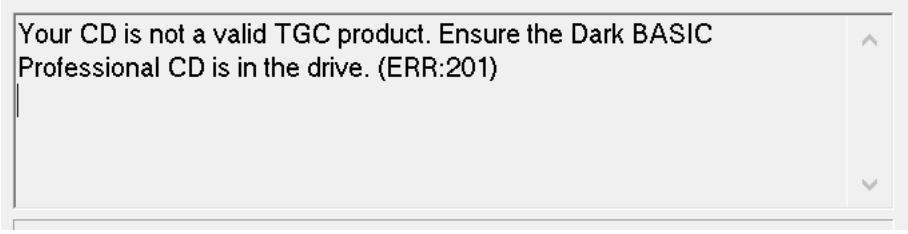
2. In this lab, we'll modify the program to remove the CD check logic, to allow continuing without a valid CD.
3. We will need to make modifications to the program, so, as always, make a backup of the original:
   - Navigate to the Dark-Basic-Pro program folder.
   - Make a backup copy of TGCOnline.exe, called TGCOnline.exe.orig. This will be the original file to restore from, in case we break the program in the process of modification.
4. At this point, we might proceed in a few different directions to dissect the target. Some options are:
   - *Dynamically*. We can reverse engineer the application by running it, typically under a debugger, and watching how it works. This allows us to set breakpoints, trace execution, and modify registers and memory, all on-the-fly, and is often the best way to experiment with making program changes.
   - *Statically*. We can reverse engineer the application without actually running it. This can be useful in many situations, such as: you may lack the right environment for running the target (for example, if reverse engineering your router firmware on your PC); you may have concerns about unknown side effects of running the program, or the efficacy of your sandboxing (for example, the program may reach out to a remote server); the program may be known malware (of course, this is one of many reasons to run in a virtual machine); the program may employ anti-debugging approaches that make dynamic analysis more difficult that static analysis; or, most commonly, you may just want to get a better high level oversight into what the program does, before you dive into dynamic analysis.

ISBN: 139419988

5. Neither way is wrong, and both skills are useful in different situations.
6. Let's start with dynamic analysis to get a feel for program modification, and we'll follow up with static analysis once we get our bearings.

Task 1

Use the information you've learned on Immunity, find a way to skip the CD check.



Tips:

- Be patient when running the program under the debugger, the instrumentation process (of any debugger) can slow execution down significantly.
- The target program may render itself on top of other windows. Move it to the side before pausing execution, so that it won't be in the way of your debugger.
- Pause execution at any time to examine the program state or make a change to the program.
- Use Immunity's patching features to modify the executable code to try to remove the CD check. You can restart the program to test your change, or save the modified executable to see if it worked.
- Remember to right-click an instruction or press 'space' on an instruction to modify it.
- Don't be afraid to try different things – trial and error is a fundamental skill here.
- If you've broken the program with an unsuccessful patch, copy from your original backup to try again.

Hints:

- Hints are available at the end of this document. Try the challenge without using hints first, you will encounter and overcome many more of the inherent challenges of the tooling and reversing process this way.

Task 2

Once you've circumvented the CD check, you will see that the program still attempts to contact the registration server. Using a similar approach, remove the server communication entirely.

Task 3

Repeat Task 1, this time using only static analysis. That is, using the tools and features in Immunity, how would you locate the correct patching location without ever running the program?

ISBN: 139419988

Optional Bonus Tasks

- Various entities sometimes rebrand software with their own logos. When running the target, notice the author's logo on the left side of the UI. Use Resource Hacker to replace the logo with an image of your own. Repackage the modified program into a new executable.
- There are always multiple ways to patch a program. Find a different location, or find a different patch in the same location, to bypass the CD check.
- Devise a patch to circumvent the registration screen entirely (that is, proceed directly to unlocking the "Finish" button, without entering any email information).
- Without patching the program, circumvent the CD checks (that is, use reverse engineering to devise a way to create a fake CD or directory that will pass the program's CD checks).

Task 1 Hints

Tip: Use hints sequentially; once you've unblocked yourself, try to continue without using the subsequent hints.

- Explore the program without a debugger. What does the program do if a valid CD is not present? Can you use any of this information (window names, text, etc.) to locate this code?
- While execution is paused in the debugger, use the memory window to search for the string "is not a valid". How could you determine when the program is using this string?
- In the memory window, right click on the first byte of the "Your CD is not a valid" string, and set a hardware breakpoint on memory accesses here. In general, hardware breakpoints can be a good choice for memory accesses.
- With the breakpoint set, press the "restart program" button to restart your program from the beginning. You may need to press the "run program" button in the debugger several times before you can interact with the target application again. Press the "next" buttons in the target application to see if you can trigger the breakpoint. When done correctly, the program should break around the following code:

```
007B59C6  . 53              PUSH EBX
007B59C7  . 56              PUSH ESI
007B59C8  . E8 13E5FFFF     CALL TGCOnlin.007B3EE0
007B59CD  . 83C4 14         ADD ESP,14
007B59D0  . 8985 BCFCFFFF   MOV DWORD PTR SS:[EBP-344],EAX
007B59D6  . 3BD8            CMP EBX,EAX
007B59D8  . 0F84 B2000000   JE TGCOnlin.007B5A90
007B59DE  . 8B15 98429B00   MOV EDX,DWORD PTR DS:[9B4298]
007B59E4  . B9 A01D9900     MOV ECX,TGCOnlin.00991DA0
007B59E9  . 8DA424 000000(  LEA ESP,DWORD PTR SS:[ESP]
007B59F0  > 8A01            MOV AL,BYTE PTR DS:[ECX]
007B59F2  . 8D49 01         LEA ECX,DWORD PTR DS:[ECX+1]
007B59F5  . 8802            MOV BYTE PTR DS:[EDX],AL
007B59F7  . 8D52 01         LEA EDX,DWORD PTR DS:[EDX+1]
```

- At the highlighted line, the program is loading the "invalid CD" string, which means it has already determined that we failed the CD check. Which instructions above might have modified the path the program takes? Which ones could we modify to change the program's decision?
- Use the debugger to modify the above "je" (jump if equal) instruction to either "jne" (jump if not equal) or "jmp" (jump always).
- Rerun the program in the debugger to check your patch, and save your patched program (right click assembly in the debugger, "copy to executable", "all modifications").

ISBN: 139419988