Lab 04 – JOP with WAVread

**Total points**: 20

**Lab Files:** You are supplied with a binary, **wavread2.exe** and instructions on how to use the wavread for its intended purpose. This is a slightly different version of the wavread.exe binary.

* *For this lab, use the Windows 7 VM provided by Dr. Brizendine in the IA Lab. No others are acceptable, unless the addresses are the same. For instance, you may have another Windows 7 that is similar. This allows the exploit to be carefully tested with the Python that you submit.*

**Preface:**

Your target is a vulnerable application that reads, processes, and provides information on a supplied WAV file. This is the same binary that you performed ROP on, but now you are doing JOP instead. Your task is similar: find a way to bypass DEP protections by calling Windows API functions of your choosing. You might try a different method than what you did previously, although you are welcome to do the same. Now that you have figured out how this binary works and how to exploit with ROP, doing JOP will be considerably easier, as you can use the same methods of supplying input, etc.

It is recommended to use the JOP ROCKET in order to find gadgets. JOP ROCKET is installed in the C:\ROCKET directory. Using a debugger will be critical. It will, of course, be very helpful to step through the exploit with a debugger in order to determine whether the exploit is behaving as intended.

For this lab, your requirement is to use at least one XOR and overwrite—even if this is not necessary. Use this to overwrite one dummy value on the stack. **You do not need to use dummy values for all stack values, and in fact this would be redundant, but you will use at least one XOR and overwrite to demonstrate mastery of this technique. The overwrite technique you use is at your discretion.**

A stub of a script is provided to get you started with part of the exploit:

|  |
| --- |
| **#!/usr/bin/python**  **import os**  **import sys**  **import struct**  **#these values are just examples and may change or not exist in the real exploit**  **offset = 1**  **padding = “\x41” \* offset**  **things = “\90” \* 50**  **#struct.pack(‘<L’,…) will account for Little-Endianness for you.**  **anAddress = struct.pack(‘<L’, 0x12345678)**  **payload = padding + things + anAddress**  **binary\_file = open(“myfilename.extension”, “w”) #create and write file**  **binary\_file.write(payload)**  **binary\_file.close()** |

**Part 1 – Bypassing DEP with JOP**

* Make sure not to disable DEP, as your goal is to bypass it.
* Look at where other input may be. This utility reads and performs operations on a .WAV file. Where in memory does the .WAV go? Can you find it? Is it suitable as a means of supplying payload? If you were to supply a malformed .WAV file, how would you approach this? Research and investigate the .WAV file format. What does a legit WAV file look like? A basic stub of a Python script is supplied to help get you started with creating a binary file.
* Figure out how to bypass DEP with JOP utilizing a Windows API of your choosing.
* **Note: The maximum points available for this lab is 10 points if you are unable to set up the stack successfully with JOP. Submissions without the Python script will not be graded.**

* **[18 points]** Provide a detailed narrative that details your JOP chain. At a minimum, discuss:
  + **[2 points]** How did you perform the stack pivot? Is this reliable? How did you begin execution of your JOP chain? How did you set up the gadgets for JOP?
  + **[2 points]** Where in memory will you place your JOP gadgets? Why? How did you get them there? Are you able to reach them and use them here? Why or why not?
  + **[5 points]** What technique does your JOP chain use to bypass DEP. Show that it works. **Include the Python script showing that this works in the IA LAB. Make sure your script is well commented to indicate the reason why you do certain tasks. Do not feel the need to comment every line. Each JOP instruction should be apparent on each line, if it is a JOP gadget—do this with comments.**
  + **[5 points]** Walk through how you set up the stack for your Windows API. Did you get the correct values in there? Did they execute as desired? Include a screenshot showing your values on the stack inside WinDbg, Immunity, or x64dbg. Indicate which are which arguments.
  + **[4 points]** Provide a detailed account of what you are doing inside the script, walking through the different elements of the script. What gave you problems? How did you overcome them? You may copy and paste excerpts of script with comments if appropriate.

**Part 2 - Writing Shellcode**

Now it’s time to create your own shellcode! You may use any shellcode of your choosing in this lab, either created by yourself or supplied by a tool. You may reuse previous shellcode. The goal here is simply to get it to work.

* **[2 points]** Show that your shellcode is ready to be executed after your JOP chain completes. Describe what your shellcode is and what it does. Are you able to execute your shellcode successfully? Include a screenshot showing that execution is successful. *Note: No points awarded if shellcode is not successful after the JOP chain.*

**Deliverable**

* A Word document or PDF with requested answers to the previous tasks.
* Python script -- Provide an exploit. Utilize the provided VM. Show a screenshot that it was effective and include the Python script you used, as it will be tested.