Lab 05 – ROP with Vet APP

**Total points**: 20

**Lab Files:** You are supplied with two binaries for veterinary management software, vetHost.exe and vetClient.exe. There are instructions on how to use this vet software for its intended purpose. This is a network-based app that permits different clients to remotely contact the host and provide information on the patients, creating records, etc.

* *For this lab, use the Windows 7 VM provided in the IA LAB. The JOP ROCKET will be installed, and you will not have to worry about installing dependencies. If you wish to use your own VM, you may do so, but it must run in the Windows 7 VM in the IA Lab, as* ***I will be running the exploit for every submission.***

**Preface:**

Your target as before is to bypass DEP via Windows API functions using ROP. Unlike with the previous tasks, this binary is locked down with full ASLR, so unless you overcome ASLR, you won’t get very far.

This is a network-based utility that has a UAF. This UAF can be used to provide a memory leak. This memory disclosure can be used to bypass ASLR. You will need to firstly discover the UAF. Then, you will need to figure out how to use it to get an appropriate disclosure to bypass ASLR.

You may want to observe a little on Wireshark to look at commands that are sent, what that looks like, so that you can replicate those if need be in Python. You don’t need to observe too much that is particularly useful in revealing the exploits, but in terms of mechanics of how to interact with the server via the client from a Python script, this is basic info that you will want. Observing through Wireshark or other means will not help you find gadgets or an ASLR.

The first step is up to you – figure out how to trigger a crash or figure out how the UAF works and where it happens. You should find the UaF through interacting with the program and looking for aberrant behavior. It also could be possible to trace through functionality in IDA/Ghidra or a debugger to look for something that is used after it is freed, to maybe give you an area to focus on. Even if you find where it is the program, you still have to see where/how it manifests itself to the user and how you can utilize that. You will need to take what you observe and work it into your script. That is, figure out what causes the memory disclosure, find a way to isolate and extra that through a Python script. Knowing what the UAF is and how to trigger it would make that easier to find. The only output from UaF that you are after is what is displayed to user or sent through the network (and what you want will also be displayed to user).

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

**Part 1 - Finding the Exploit**

* Run the program and understand how it works and how you can interact with it. Think about the flow of user-supplied data - trace those functions to find where the buffer overflow occurs.
* Once you’ve identified the **vulnerable function**, determine how large your payload needs to be in order to exploit.
* Can you cause a buffer overflow to occur and overtake EIP?
* **[3 points]** Provide details about the vulnerability such as how you found it and how you were able to exploit it. You should be able to identify exactly where in IDA/Ghidra the vulnerable function is. Also provide evidence via a debugger that you had control of EIP at this point in the lab.

**Part 2 - Finding the Memory Disclosure**

* Run the program and interact with it; look at the disassembly if need be. See if you can find the UAF. Once you find the UAF, use it and get your memory disclosure?
* What is the memory disclosure? What is the distance to the base, i.e. the offset? Show your work and explain. **[5 points]**
* Once you have identified the UAF, can you find it and **explain the cause**? What **caused the UAF**?Depending on how you find it—whether through looking at disassembly (static analysis) or just interacting with it, you may or may not need to use Gflags to try to discover the root cause. You should be able to identify exactly 1. where in IDA/ Ghidra that the UAF occurs and 2. how it made use of for a memory disclosure. (This is more than one location.) What can you do to make this memory disclosure **useful**? Explain. **[2 points]**

**Part 3 – Bypassing DEP with ROP**

* This step is **not possible** without ASLR bypass. Do not submit hypothetical, “what if there were no ASLR”-code. Make sure ASLR is bypassed via memory disclosure before embarking on this part.
* Figure out how to bypass DEP with ROP. You may need to use an overwrite.
* **[8 points]** Provide a detailed narrative that details your ROP chain. At a minimum, discuss:
  + **[1 points]** How did you begin execution of your ROP chain?
  + **[7 points]** What technique does your ROP chain use to bypass DEP. Show that it works. **Include the Python script showing that this works in the Windows 7 IA lab, so that your script can be tested in the same environment. Make sure to have detailed comments in the script, explaining functionality.** Include a narrative that walks through and explains what you did and how you did it, any obstacles, anything noteworthy. You do not need to explain every detail. A screenshot is not a narrative.

**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. If you write your own, **include the .ASM**.

* **[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 ROP chain.*** Include screenshot.
* **[Extra Credit –** **up to 3 points]** Do something special and **complex**, relevant and **useful** to exploitation here—make sure this differs from anything you have done before. Maybe writer a decoder and user it with your shellcode, or maybe you don’t use the skeleton, but make your **own from scratch**! There are various possibilities to get up to 3 points extra credit; the amount will be determined by what is submitted. 0 points if using someone else’s shellcode. Please indicate that you are attempting extra credit in your submission, by putting it in a section called **Extra Credit Shellcode.** This

**Deliverable**

* A Word document or PDF with requested answers to the previous tasks. Page limit maximum: 13 pages. Use appropriate screenshots and good, specific detail, but be concise. Be professional; this is a report.
* 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. Python script should be well commented / annotated, with comments for each ROP gadget telling what it is doing. You do not need to comment every single line, but the important things should have comments where appropriate.