Debugging & Exploiting Test.exe

Exploiting the Test.exe Application

In this Lab we want to write an exploit to achieve arbitrary code execution using a vulnerability in the Test.exe application. This type of exploit is considered a local exploit, since you can only speak to the vulnerable application through the running system. The application is vulnerable to a memory corruption (buffer overflow) which is found when loading a long string. Therefore, in order to exploit the application, we need to create a long string with malicious code to use and achieve our exploitation purposes.

Note: all files needed can be found here too.

Use the code below to start with (**SIZE** = the value you find during debugging).

```
import subprocess

buf = ("A"*SIZE)
print "Buffer: ", buf, " Buffer length: ", len(buf)

p1 = subprocess.Popen(["C:\Program Files (x86)\Immunity Inc\Immunity
Debugger\ImmunityDebugger.exe",
                stdin=subprocess.PIPE, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
stdout, stderr = p1.communicate()
print stdout, stderr
```

Shellcode to Open Calculator (113 bytes):

```
shellcode=("\xdd\xc1\xb8\x7d\x5f\x6b\x07\xd9\x74\x24\xf4\x5d\x29\xc9\xb1"
"\x13\x31\x45\x1a\x83\xed\xfc\x03\x45\x16\xe2\x88\xdc\x8f\xfb"
"\x43\x31\x1d\x6c\xc0\xd4\xce\x0f\x52\x4e\x5d\x81\x3e\xfb\x13"
"\x11\x35\x8d\xdf\xda\x3f\x7e\x4d\x57\x8f\xf5\xf3\x70\x64\x56"
"\x30\x0a\x26\x52\x30\x87\xa2\xbb\xe0\x96\xb4\x4f\xb4\x87\x6c"
"\x5f\x82\x94\x7b\x1d\xae\x49\x02\x9d\x29\xc5\x6d\xb0\x70\x9c"
"\x9d\xc7\x0f\x41\x41\xd9\x11\x7e\x45\x77\x12\x56")
```

Task #1: Crashing the Application & Controlling EIP

Follow the recorded videos to crash the application using a string of "**A**"s. Use the debugging videos to help you walk through this process.

Deliverable #1: Did the application crash? Provide proof (screenshot).

Deliverable #2: Could you see any "A"s in any of the registers? Where? Explain your findings with screenshots.

Deliverable #3: Adjust the payload you send the application to show 4 "B"s in EIP. Provide a screenshot for proof.

Task #2: Finding a Jumping Address

Find a jumping address that can be used to jump to where ESP is pointing to. Use the recorded videos on canvas to help you walk through this process.

Deliverable #4: What is the address used?

Deliverable #5: Do you have an idea why we cannot use any of those marked in RED?

Deliverable #6: Adjust your code so this time you stop on the jump address. Provide proof of that.

Task #3: Running Arbitrary Code

Everything is working perfectly as we want. Now let us adjust our exploit to run the Windows Calculator that proves our exploitation was successful.

Add the shellcode found at the beginning of this lab document to your code and do the proper adjustments to proceed. Do not worry about how this code was generated, we will come to that, plus many others later, for now we want to complete our PoC.

One thing I would recommend before running your code, is to add some No Operation (0x90) instructions before the buffer holding the payload is reached. Sometimes this is best for alignment purposes and to make sure we jump into the right landing zone to run our injected code.

Deliverable #7: Provide proof of running the calculator from within the debugger and even without a debugger with the final exploit code used.

Task #4: Lab Reflection

Deliverable #8: Please provide a reflection on what you learned from this lab and any other feedback you would like to add or have me go over during the next class.

Congrats on your first memory corruption (BoF) exploitation...