Journeyman RE/VR Practice

Monday, May 11, 2020 3:51 PM

Using gcc/gdb

Force segmentation fault

```
#include <stdio.h>
#include <string.h>

int main()
{
    int array[5];
    int i;

    for (i = 0; i <= 255; i++)
    {
        array[i] = 10;
    }
}
```

```
c:\> gcc overflow.c -o overflow.exe
c:\> gdb overflow.exe
    (gdb) start
    (gdb) continue
        • "Segmentation Fault"
        • "0x00401417 in main()"
    (gdb) continue
        • "Segmentation Fault"
        • "0x0000000a in ?? ()"
            ▲ 0x0a = 10
```

How function + returns work

```
void function(int a, int b)
{
  int array[5];
}

main()
{
  function(1,2);

  printf("This is where the return address points");
}
```

```
c:\> gcc -mpreferred-stack-boundry=2 -ggdb overflow.c -o overflow.exe
c:\> gdb overflow.exe
    (gdb) disas main
    (gdb) disas function
    * RET (EIP) is pushed onto
      stack before function() call
    * perform function then returns.
```

Overflowing Buffer on Stack

```
void return_input (void)
{
  char array[30];

  gets (array);
  printf("%s\n", array);
}

main()
{
  return_input();

  return 0;
}
```

// it seems we are looking for a called function that ends with an exploitable call... such as printf when it prints an overflowed buffer

```
c:\> gcc -mpreferred-stack-boundry=2 -ggdb overflow.c -o overflow.exe
c:\> gdb overflow.exe
    (gdb) disas return_input
    (gdb) break *0x401411c      // gets ()
    (gdb) break *0x401427       // puts ()
    (gdb) run
    (gdb) disas main
          // find addy after <return_input>
            because this is RET ADDY (0x00401436)
    (gdb) x/20x $esp
          0x61ff0c: ....  ....  ....  ....
          0x61ff1c: ....  ....  ....  ....
          0x61ff2c: ....  ....  0x00401436  0x0061ff94   RET ADDY  EBP
          0x61ff3c: ....  ....  ....  ....
          0x61ff4c: ....  ....  ....  ....
    (gdb) continue
          > AAAAAAAAAABBBBBBBBBBccccccccccDDDDDDDDDD
            array [30]                    pppppss!!
          * Breakpoint 2
    (gdb) x/20x 0x0061FF0C
          0x61ff0c: ....  ....  ....  ....
          0x61ff1c: ....  ....  ....  ....
          0x61ff2c: ....  ....  0x44444444  0x00004444   RET ADDY  EBP
          0x61ff3c: ....  ....  ....  ....
          0x61ff4c: ....  ....  ....  ....
    (gdb) x/1i $eip
    (gdb) stepi     // continue
          0x44444444 in ?? ()
            * AAAAAAAAAABBBBBBBBBBccccccccccDDDDDDDDDD
              array [30]                  +2 EBP RET
            // we have successfully changed the return value to 0x44444444 because the buffer that was overloaded and printed was at the end of the function and right next to the EBP and RET stored values
```

Controlling EIP

```
// we are going to set EIP to "RET" as to <return_input>
(gdb) disas main
        0x00401437  call  0x401410 <return_input>
```

Programs

- VS 2019
- PyCharm
- PEStudio
- Atom
- CFF Explorer
- MinGW/gdb

GDB Commands

- Start
- Continue
- disas main
- break *0x401411c
- set disassembly-flavor intel
- skip
- quit
- run

Break

SIDEBARS

GDB w/ Live Overflow

- set disassembly-flavor intel
- break *main
- info registers
  - rip = current instruction ptr
- si (stepi) // steps into
- ni (nexti) // steps over
- <enter> to repeat last
- set $eax=0 // when you encounter a strcmp followed by a Jump If Not Zero (jne/jnz), change the test result, eax, to zero to effectively bypass the strcmp check
- c:\> objdump -d overflow.exe
- x/s 0x4006da // view string at address
  - > 0x4006da: "AAAA-ZINN-42-OK"
  - // Smashing the Stack for Fun and Profit
- info proc mappings
- del // delete breakpoints
- define hook-stop

STACK FRAME SETUP

```
libc_start_main:
    call  <main>                  // push next addy (eip) to stack
    mov   DWORD PTR [esp], eax    // jumped (return to later)
    ...
main:
    push  ebp                     // push current location of ebp to stack
    mov   ebp, esp                // overwrites ebp with esp to be base of frame
    and   esp, 0xfffffff0         // makes it pretty (need more info)
    sub   esp, 0x60               // Stack Ptr bounces away to create stack space
    mov   DWORD PTR [esp+0x5c], 0x0   // puts 0 in stack ≥ 64 chars
    ...
    leave                         // pops (destroys) stack
    ret                           // pops return addy to eip
```

RANDOM NOTES

- return_input = 0x401410

  \x10\x14\x40\x00

py >>> os.system("test.exe ABCD")

* use OLLYDBG!

py >> os.system("OLLYDBG.EXE overflow.exe ABCD\x10\x14\x40\x00")

* USE CFF EXPLORER FOR IAT!
* USE OLLY Exec Mod E to find Kernel.dll
- Kernel "jmp ESP" command =

  0x 75 E0 9A 57 76 D9 F5 47

  \x57\x9A\xE0\x75

- ntdll "jmp esp" command =