

Arc and heap overflow attacks

1. What makes a return-into-libc/arc-injection attack attractive to an attacker?
2. Where does the code come from that an attacker executes in a return-into-libc attack?
3. What kinds of attack does NX (No eXecute) hardware-support protect against?
4. What kinds of attack does address space randomisation protect against?
5. What kinds of attack does StackGuard protect against?
6. Given the following addresses, draw a diagram of the stack that implements a return-into-libc attack that raises privileges prior to launching a shell:

EBP of vulnerable frame	0xbfff0000
system	0x40abcdef
setresuid	0x42badcfe
leave / ret sequence	0x083412ff
"/bin/sh"	0xdeadbeef

7. Where does the overflow occur in a heap overflow attack?
8. What do `malloc`, `calloc`, `realloc` and `free` do?
9. Describe the problem of heap fragmentation.
10. The DL heap manager does not like adjacent free chunks of memory on the heap. Why not? What does it do about them?
11. Does a call to `malloc(0)` consume memory?
12. What does a heap overflow vulnerability potentially allow an attacker to do?
13. Why would you zero out a chunk of memory before freeing it?
14. In a heap overflow attack a free chunk is engineered. How?
15. Describe how in a heap overflow attack the DL heap manager tricked into unlinking the fake free chunk.
16. What exactly does unlinking a fake free chunk allow an attacker to do?