buffer overflow exploits - Why is the shellcode put before the return address

Ask Question

Asked 7 years, 6 months ago Active 6 years, 10 months ago Viewed 4k times

2

The code I'm reffering to is found here: Link to code

I read that the buffer overflow exploit uses a buffer that looks something like this:

| NOP SLED | SHELLCODE | REPEATED RETURN ADDRESS |

From what I understand the exploit happens when the buffer is put onto the stack as a function parameter and overwrites the function's return address. I also understand that the repeated return address points to the NOP sled in the same buffer on the stack.

What I don't understand are the following:

1. Why does the return address have to point to the shellcode in the same buffer? Why not have the reapeated return addresses point to another part of the memory where the NOP sled and the Shellcode can be found?
2. How is the return address on the buffer perfectly aligned with the original one so the "ret" command will read the correct address and not read it from the middle for example.

c security x86 buffer-overflow exploit

share improve this question follow

edited May 23 '17 at 11:58
Community ♦
1 ● 1

asked May 28 '13 at 10:02
Shookie
4,525 ● 6 ● 28 ● 55

add a comment

1 Answer

Active Oldest Votes

3

1. The return address doesn't have to point to code in the same buffer, it is just often easier to do it this way. If you can put the shell code in and return address into the same buffer then this is the simplest. If the buffer that can be overflowed is too small to fit the shell code, it is feasible to put the shell code into another buffer and then jump to that when the vulnerable buffer is overflowed.

   Also, protections such as Data Execution Prevention or (NX) prevent code being executed from a stack. In this case, techniques such as Return-Oriented Programming can be used to circumvent DEP. This technique involves using legitimate, executable code segments to run code the attacker wants to.

2. This can be tricky and may require some fiddling around with the payload. *Usually* the start of a buffer is at an address that is `word` aligned. In this case, ensuring the return address is correctly aligned means writing a buffer that is a multiple of the CPU word (4 bytes for 32-bit machines, 8 bytes for 64-bit). If the buffer is not word aligned, then an attacker may just experiment by adding or removing byte at a time until he thinks it is.

The reason it is simpler to do everything in one buffer is because not much will change between the injection of the shell code and the jumping to the newly modified return address. At the point of attack, it is very unlikely that an attacker can reference memory of another process, so we must look at buffers in process.

Putting shell code into a different buffer requires the attacker to understand how long the buffer will stay in place. Do different function calls cause one of the buffers to be deallocated? Is one of the buffers on the heap instead of the stack? So long as your single buffer is large enough, it is much simpler to put your NOP sled and shell code near the start and then just fill the rest with the return address. Compared to finding one buffer to populate with shell code and another to populate with the address of the previous buffer. Some shell code may also reference the stack pointer which means it needs to be set correctly.

share improve this answer follow

edited Jun 20 at 9:12
Community ♦
1 ● 1

answered May 28 '13 at 12:03
Steve
6,749 ● 2 ● 25 ● 48

Thanks @Steve! I don't get why it's the simplest method though. Why is building the buffer this way simpler than having a buffer full of the return address for another buffer with the shellcode? Someone told me I can't have the return address point to another buffer with a shellcode because it's not the same process or something of that sort. Does that make sense? – Shookie May 28 '13 at 13:13

@Shookie I've added a bit more to the answer, hopefully that helps :) – Steve May 28 '13 at 13:25

Thanks! One more question though: What happens after the shell code is executed? Do I need to supply a ret instruction at the end of the shellcode so that it won't read garbage? If I do, won't it return to the wrong function (since I've overwritten the previous return address)? – Shookie May 28 '13 at 13:32

It depends on what you want to do. If you don't care about the application crashing, you can just do nothing and let it crash. Otherwise you can try and return to the previous function. – Steve May 28 '13 at 13:56

But if I've overwritten the previous function's return address how would I do that? – Shookie May 28 '13 at 14:32

show 3 more comments

Your Answer

B I

Sign up or log in

Sign up using Google

Sign up using Facebook

Sign up using Email and Password

Post as a guest

Name

Email

Required, but never shown

Post Your Answer

By clicking "Post Your Answer", you agree to our terms of service, privacy policy and cookie policy

Not the answer you're looking for? Browse other questions tagged c security x86 buffer-overflow exploit or ask your own question.