I modified your program a little bit to make it more illustrative:

#include <iostream>

int main( void )

{

int authentication = 0;

char cUsername[ 10 ];

char cPassword[ 10 ];

std::cout << "Username: ";

std::cin >> cUsername;

std::cout << "Pass: ";

std::cin >> cPassword;

if( std::strcmp( cUsername, "admin" ) == 0 && std::strcmp( cPassword, "adminpass" ) == 0 )

{

authentication = 1;

}

if( authentication )

{

std::cout << "Access granted\n";

std::cout << ( char )authentication;

}

else

{

std::cout << "Wrong username and password\n";

}

return ( 0 );

}

I compiled it with x64 compiler command-line MS compiler, no optimizations. So now we have an exe that we want to "hack". We load the program with WinDbg (really good debugger) and take a look at the disassembly (notice, I've supplied full debug info, for clarity):

00000001`3f1f1710 4883ec68 sub rsp,68h

00000001`3f1f1714 488b0515db0300 mov rax,qword ptr [Prototype\_Console!\_\_security\_cookie (00000001`3f22f230)]

00000001`3f1f171b 4833c4 xor rax,rsp

00000001`3f1f171e 4889442450 mov qword ptr [rsp+50h],rax

00000001`3f1f1723 c744243800000000 mov dword ptr [rsp+38h],0 // This gives us address of "authentication" on stack.

00000001`3f1f172b 488d156e1c0300 lea rdx,[Prototype\_Console!std::\_Iosb<int>::end+0x78 (00000001`3f2233a0)]

00000001`3f1f1732 488d0d47f00300 lea rcx,[Prototype\_Console!std::cout (00000001`3f230780)]

00000001`3f1f1739 e8fdf9ffff call Prototype\_Console!ILT+310(??$?6U?$char\_traitsDstdstdYAAEAV?$basic\_ostreamDU?$char\_traitsDstd (00000001`3f1f113b)

00000001`3f1f173e 488d542428 lea rdx,[rsp+28h] // This gives us address of "cUsername" on stack.

00000001`3f1f1743 488d0df6f00300 lea rcx,[Prototype\_Console!std::cin (00000001`3f230840)]

00000001`3f1f174a e823faffff call Prototype\_Console!ILT+365(??$?5DU?$char\_traitsDstdstdYAAEAV?$basic\_istreamDU?$char\_traitsDstd (00000001`3f1f1172)

00000001`3f1f174f 488d153e1c0300 lea rdx,[Prototype\_Console!std::\_Iosb<int>::end+0x6c (00000001`3f223394)]

00000001`3f1f1756 488d0d23f00300 lea rcx,[Prototype\_Console!std::cout (00000001`3f230780)]

00000001`3f1f175d e8d9f9ffff call Prototype\_Console!ILT+310(??$?6U?$char\_traitsDstdstdYAAEAV?$basic\_ostreamDU?$char\_traitsDstd (00000001`3f1f113b)

00000001`3f1f1762 488d542440 lea rdx,[rsp+40h] // This gives us address of "cPassword" on stack.

00000001`3f1f1767 488d0dd2f00300 lea rcx,[Prototype\_Console!std::cin (00000001`3f230840)]

00000001`3f1f176e e8fff9ffff call Prototype\_Console!ILT+365(??$?5DU?$char\_traitsDstdstdYAAEAV?$basic\_istreamDU?$char\_traitsDstd (00000001`3f1f1172)

00000001`3f1f1773 488d15321c0300 lea rdx,[Prototype\_Console!std::\_Iosb<int>::end+0x84 (00000001`3f2233ac)]

00000001`3f1f177a 488d4c2428 lea rcx,[rsp+28h]

00000001`3f1f177f e86c420000 call Prototype\_Console!strcmp (00000001`3f1f59f0)

00000001`3f1f1784 85c0 test eax,eax

00000001`3f1f1786 751d jne Prototype\_Console!main+0x95 (00000001`3f1f17a5)

00000001`3f1f1788 488d15291c0300 lea rdx,[Prototype\_Console!std::\_Iosb<int>::end+0x90 (00000001`3f2233b8)]

00000001`3f1f178f 488d4c2440 lea rcx,[rsp+40h]

00000001`3f1f1794 e857420000 call Prototype\_Console!strcmp (00000001`3f1f59f0)

00000001`3f1f1799 85c0 test eax,eax

00000001`3f1f179b 7508 jne Prototype\_Console!main+0x95 (00000001`3f1f17a5)

00000001`3f1f179d c744243801000000 mov dword ptr [rsp+38h],1

00000001`3f1f17a5 837c243800 cmp dword ptr [rsp+38h],0

00000001`3f1f17aa 7426 je Prototype\_Console!main+0xc2 (00000001`3f1f17d2)

00000001`3f1f17ac 488d15151c0300 lea rdx,[Prototype\_Console!std::\_Iosb<int>::end+0xa0 (00000001`3f2233c8)]

00000001`3f1f17b3 488d0dc6ef0300 lea rcx,[Prototype\_Console!std::cout (00000001`3f230780)]

00000001`3f1f17ba e87cf9ffff call Prototype\_Console!ILT+310(??$?6U?$char\_traitsDstdstdYAAEAV?$basic\_ostreamDU?$char\_traitsDstd (00000001`3f1f113b)

00000001`3f1f17bf 0fb6542438 movzx edx,byte ptr [rsp+38h]

00000001`3f1f17c4 488d0db5ef0300 lea rcx,[Prototype\_Console!std::cout (00000001`3f230780)]

00000001`3f1f17cb e825f9ffff call Prototype\_Console!ILT+240(??$?6U?$char\_traitsDstdstdYAAEAV?$basic\_ostreamDU?$char\_traitsDstd (00000001`3f1f10f5)

00000001`3f1f17d0 eb13 jmp Prototype\_Console!main+0xd5 (00000001`3f1f17e5)

00000001`3f1f17d2 488d15ff1b0300 lea rdx,[Prototype\_Console!std::\_Iosb<int>::end+0xb0 (00000001`3f2233d8)]

00000001`3f1f17d9 488d0da0ef0300 lea rcx,[Prototype\_Console!std::cout (00000001`3f230780)]

00000001`3f1f17e0 e856f9ffff call Prototype\_Console!ILT+310(??$?6U?$char\_traitsDstdstdYAAEAV?$basic\_ostreamDU?$char\_traitsDstd (00000001`3f1f113b)

00000001`3f1f17e5 33c0 xor eax,eax

00000001`3f1f17e7 488b4c2450 mov rcx,qword ptr [rsp+50h]

00000001`3f1f17ec 4833cc xor rcx,rsp

00000001`3f1f17ef e8bc420000 call Prototype\_Console!\_\_security\_check\_cookie (00000001`3f1f5ab0)

00000001`3f1f17f4 4883c468 add rsp,68h

00000001`3f1f17f8 c3 ret

Now, since we know how x64 stack works we can start "hacking". RSP is stack pointer, function stack is addresses above RSP value (stack grows into smaller addresses). So, we see that RSP+28h is where cUsername, RSP+38h is authentication, and RSP+40h is cPassword, where 28h, 38h and 40h are hexadecimal offsets. Here is little image to illustrate:

-----> old RSP value // Stack frame of caller of `main` is above, stack frame of main is below

16 bytes of

"cPassword"

+40h

8 bytes of "authentication"

+38h

16 bytes of

"cUsername"

+28h

-----> RSP value = old RSP-68h

What do we see from here? We see that compiler aligned data on 8 byte boundary: for example, we asked to allocate 10 bytes for cUsername, but we got 16 bytes - x64 bit stack is aligned on 8-byte boundary, naturally. That means in order to write into authentication we need to write into cUsername MORE that 16 bytes (symbols). Notice also, that compiler put cPassword higher that authentication - we cannot overwrite authentication using cPassword, only cUsername.

So now we run our program and input Username: 0123456789abcdef1. 0123456789abcdef = 16 bytes, the next 1 is going to be put into lower byte of authentication - good enough for us:

Username: 0123456789abcdef1

Pass: whatever

Access granted

1