Link to Homework : https://elearn.ucr.edu/courses/67226/assignments/433460

Please briefly answer the following questions:

Q1) From the stack buffer overflow paper, what is NOP sledge and why is it useful?

NOP sledge is adding a bunch of NOP statements at the start and end of our code. This is done so that we do not need an exact return address to the start of code. This way the return address could point to any NOP statement and then will be directed to our code. The NOP sledge is added to the end of code so that in case we use a shell code that pushes on stack, we do not overwrite our own code.

Q2) From the stack buffer overflow paper, what is the simplest way to find potential buffer overflow Vulnerabilities?

Search for unsafe functions like strcpy and gets. These functions generally do not have a check for buffer length and can easily be exploited for buffer overflow.

Q3) From the StackGuard paper, why did it mitigate the easy-to-guess canary problem?

The author mitigated the easy-to-guess canary problem as any attacker could guess the canary and add it to the attack string and use it to overwrite the canary. By doing this, the attacker would be able to change the return address and the stackguard would not detect it as the value of canary remained the same. As a solution the paper implements a random canary that is different for each program and each function.

Q4) From the StackGuard paper, MemGuard provides better protection than StackGuard, why it is not widely adopted like StackGuard?

Due to performance issues. MemGuard uses extra computations and memory for using quasi-invariant values to check if the read only value is being changed or not. This Is a reason it is used alongside the canary variant which is weaker but faster.

Q5) From the ROP (return-oriented programming) paper, why does the Galileo algorithm scans backwards instead of forwards?

Because it is simpler to scan backwards from an already found sequence than to disassemble forward from every possible location in hope of finding the sequence.

Q6) ROPgadget is a tool to facilitate ROP exploits, try its --ropchain option on a libc binary (e.g., /lib/x86_64-linux-gnu/libc.so.6) and explain how the found shellcode works, referring to Section 4 of the ROP paper.

Used ROPgadget on /lib32/libc.so.6 and got 112070 gadgets.
ROP chain was possible.

Shellcode binary:
b'y\x14\x02\x00``"\x00\xc4q\x03\x00/bin\xb6\t\n\x00AAAAAAAAy\x14\x02\x00d`"\x00\xc4q\x03\x00//sh\xb6\t\n\x00AAAAAAAAy\x14\x02\x00h`"\x00\xba\xd7\x03\x00AAAAAAAAAAAAAAAA\xb6\t\n\x00AAAAAAAA_\xbf\x02\x00``"\x00\xc3q\x03\x00h`"\x00AAAA\xc4q\x03\x00h`"\x00PM\x03\x00~\xf6\x02\x00~\xf6\x02\x00~\xf6\x02\x00~\xf6\x02\x00~\xf6\x02\x00~\xf6\x02\x00~\xf6\x02\x00~\xf6\x02\x00~\xf6\x02\x00~\xf6\x02\x00~\xf6\x02\x00\x85u\x03\x00'

Step 1: pop esi and return
Step 2: Read Data
Step 3: pop edx and return
Step 4: append '/bin' to the bytes array. This will be used as input
Step 5: move dword from esi to edx and pop ebx and pop esi and return
Step 6: add padding for 8 bytes
Step 7: pop esi and pop edx and return
Step 8: Read data +4 bytes
Step 9: Pop edx and return
Step 10: append '/sh' to the bytes array. This will be used as input
Step 11: move dword from esi to edx and pop ebx and pop esi and return
Step 12: add padding for 8 bytes
Step 13: pop esi and return
Step 14: Read data+8 bytes
Step 15: xor edx,edx and pop ebx and pop esi and pop edi and pop ebp and return
Step 16: Add padding for 16 bytes
Step 17: move dword from esi to edx, pop ebx, pop esi and return
Step 18: Add padding 8 bytes
Step 19: pop ebx and return
Step 20: pop ecx and pop edx and return
Step 21: Add padding for 4 bytes
Step 22: Xor eax with eax and return
Step 23: Increment eax and return for 12 bytes
Step 24: Call exit (0x80)

Q7) From the Control-flow Integrity (CFI) paper, what are the three requirements/assumptions for CFI enforcement?

1) Unique Ids - The bit pattern's chosen IDs should not be present in anywhere in the code memory
2) NonWritable Code - It is not possible for the program to modify code at runtime.
3) Non Executable data - The program should not execute data as it is code.

Q8) From the CFI paper, what is the strategy mentioned in the paper that can improve the accuracy of CFG?

- One strategy to improve precision is to add duplicate code. This will allow the program to target two different destination sets.
- Another strategy is to Refine the instrumentation. We can add more than one ID at certain destinations or ID check can only check certain bits of the destination ID instead of all the bits.