Instructions: Write your name and NetID above! The quiz contains 6 questions worth 10 points each. All questions are mandatory. Each part of every question has a short (1-4 sentences) answer that will get full credit. Longer answers are less likely to receive full credit. Write your answers concisely in the space provided below the questions. The amount of blank space doesn’t indicate the length of the answer.

The descriptions of the problems may seem long, but that is because they aim to make the problems as self-contained as possible. Don’t panic! Read the descriptions slowly and carefully. Some of the solutions have been already described in the lectures. If you have any doubt, make appropriate assumptions and clearly mention them.
Questions:

1. A student in the fabulously named COMPSCI 290-02 “Topics in Computer Science” class has decided to have a little fun with his instructor by defacing the test page set up for the instructor on a web server that is purposely vulnerable to the ShellShock bug. The student tries over and over again to deface the instructor’s page using the telnet application, as shown below, but it never works!

```
telnet 152.3.136.182 8081
GET /cs290/bmm25.html HTTP/1.1
Host: 152.3.136.182
User-Agent: () { :;}; echo UNC_fan > /home/cps_test/apache2/htdocs/cps290/bmm25.html
```

Why doesn’t this work?

**Solution:** The ShellShock bug occurs when the bash shell improperly evaluates environment variables. In order to exploit the bug, the client must induce the server to run the bash shell. In this example, the client is asking to receive a file, /cs290/bmm25.html, which the server will deliver without starting bash. What the client must do instead is request something in the /cgi/bin directory.
2. Suppose that a web site is operating as a Tor hidden service, and does not provide direct access through the public Internet. How could an attacker who knows the onion address (e.g., XYZ.onion) of the web site attempt to bring down the site, assuming that the attacker is not operating any relay nodes and cannot see any traffic sent or received by relay nodes except for traffic that the attacker directly sends to or receives from relay nodes.

**Solution:** Although the attacker cannot easily discover the network address of the server hosting the hidden web site, the attacker can still send requests to the server. So the attacker can launch a distributed denial of service (DDOS) attack against the site, or the attacker can attempt to exploit bugs such as ShellShock by sending properly crafted requests.
3. The Hewlett Packard PA-RISC processor architecture does not have explicit push and pop instructions. By convention, however, general register r30 is used as the stack pointer, and the stack grows “upwards” in the sense that as function calls are made and the stack grows, the value of r30 increases. (In contrast, on the Intel x86 architecture and many others, the stack grows downwards.)

Without going into all the messy details, when a function is called, first the arguments to the called function are pushed onto the stack, then the return address is pushed onto the stack, and then space is allocated on the stack for the local variables of the called function.

In the following C code, function main passes a pointer to a local array called input_string to function get_string. The intended purpose of function get_string is to solicit an input string from a user and put it into the buffer that it was asked to fill. Function gets is a C-library function that reads a string of arbitrary length from standard input and stores it at the location specified by its argument. Every C program begins execution by invoking function main.

Draw a picture of the stack and explain how this code is vulnerable to a buffer-overflow exploit when compiled for a PA-RISC machine.

```c
#include<stdio.h>
void get_string(char *buffer)
{
    printf("Please enter a string:\n");
    gets(buffer);
}
void main()
{
    char input_string[8];
    get_string(input_string);
}
Solution:
After the call to `get_string`, but before the call to `gets`, the stack looks as follows, with `input string`, at the bottom of the stack, stored at the lowest address.

```
return address (4-byte address, where to return to when get_string is done)
buffer       (4-byte pointer, argument to get_string)
input_string (8-byte array of characters, local variable of main)
```

When `get_string` calls `gets`, it passes it the pointer `buffer`, which points back to the array called `input_string`, which a local variable of function `main` and stored on the stack, as shown above. If the user enters more than 8 characters, the additional characters will overflow first into the pointer `buffer` on the stack, and then into the return address. (Note that the overflow actually starts if the user enters 8 characters, because `gets` always adds a byte with value 0 to the end of the string.) So an attacker can change the address at which the processor should execute code following the return from `get_string`. 
4. A physical BitCoin might be constructed as follows. On the outside, it shows a public key. On the inside is a matching private key, perhaps stored on a piece of paper. The private key cannot be revealed without physically damaging the BitCoin in a way that is visible to the casual observer. Anyone who examines a physical BitCoin can compute the address that corresponds to the coin’s public/private keypair (the address is just a 160-bit hash of the public key), and check the public ledger to see that some BitCoin value has been transferred to that address, and hence can be spent by whoever possesses the private key. The “minter” of a physical BitCoin promises that the private key has been “forgotten,” i.e., no copy of the private key exists anywhere except inside the physical BitCoin.

An enterprising minter plans a public demonstration to convince the buyers of her physical BitCoins that the private keys are stored only inside the coins. She sets up a battery-powered minting machine with no cables attached suspended by a jet-fuel powered hovercraft inside a Faraday cage, and invites observers to watch it spit out physical BitCoins as they are minted. The observers cannot see the internal workings of the minting machine. After a few hours of minting coins the machine is shut down. Then vats of acid are poured on the whole apparatus until the Faraday cage, the minting machine, and the hovercraft are reduced to liquid goo. Then a large explosion is set off, and finally the remains are fired into space by a rocket targeted at a black hole.

Can the buyers of the physical BitCoins be sure that the minter won’t later spend them? Explain.

Solution: No. Although the buyers may be convinced by the demonstration that the machine that minted the physical coins has been completely destroyed, and that it did not leak any information while it was minting the coins, there is still the possibility that the private keys were created and stored by the minter both inside the machine and on some external device before the demonstration began. The machine, then, might be spitting out coins with these pre-stored private keys. (Or the machine might be generating the private keys using a deterministic algorithm that the minter can run somewhere else before or even after the physical minting is concluded.) If the minter has the private keys, then she can spend the coins.
5. Microsoft’s BitLocker feature encrypts the hard drive (e.g., C:) that contains the Windows operating system and user data files. To decrypt the drive, the user must provide the boot loader with the password that was used as the encryption key. Microsoft allows a user to enable BitLocker on a machine without a Trusted Platform Module (TPM), but warns the user that there is a security risk in doing so, which is related to Microsoft’s Secure Boot feature. Can you explain what the risk is?

**Solution:** On a machine with a TPM, Secure Boot will check that the boot loader has been signed by Microsoft. Without a TPM, a compromised boot loader might not be detected. Since the boot loader asks the user for the decryption password, a compromised boot loader might save this password, whereas normally the password is not stored anywhere on the computer. If the computer later came into the possession of an attacker, the compromised boot loader could use the password to decrypt the disk. This attack might even be carried out over the network. In summary, Microsoft is warning the user because, without a TPM, the encryption password is at risk of being captured.
6. A common approach to creating a polymorphic virus is to encrypt the payload of each copy of the virus with a different key. Encryption tends to generate data with high entropy, meaning that the bits in a ciphertext are hard to distinguish from random bits. So perhaps any file downloaded from the web or received as an email attachment that has high entropy should be considered suspicious. Compressed files, such as .zip and .jpg files, however, also have high entropy, which at first glance suggests that categorizing files based on entropy won’t be effective. But there is a fundamental difference between compressed files and encrypted files. Explain how an anti-virus program might be able to treat the two types differently.

Solution: There are several important differences between compressed files and encrypted files, including:

- Compressed files are generally not polymorphic. Each time a file is compressed, the resulting compressed version of the file is the same. Hence, the anti-virus program can check to see if the compressed file is on a list of known threats.

- Compressed files generally conform to a well known format, such as the zip format, and can be decompressed using implementations of well-known algorithms that are already stored on the computer receiving the file. Hence, the anti-virus program can decompress the file and inspect it, if desired, without running any unknown code embedded within the file.

- Compressed files generally doesn’t contain raw executable code, such as the decryption code present in a polymorphic virus. Hence, the anti-virus program can look for the signature of a decryption routing within the file.