Problem 1 It is common in C programs to see memory allocation idioms such as

```c
struct s *sarray = malloc( nelm * sizeof(struct s) );
if (sarray == NULL) exit(EXIT_FAILURE);
/* else use sarray[0], sarray[1], etc. */
```

where the number of elements `nelm` is derived from program input.

(a) What is the problem with this idiom? Under what circumstances can the attacker cause the program to misbehave, and what can he achieve as a result? Be specific.

(b) Write a function `alloc_array` that is safe to use instead. The function should call `malloc` when it is safe to do so, and return `NULL` when it is not. The function prototype should be `void *alloc_array(size_t nelm, size_t elmsize)`.

You may assume a 32-bit system, with the following definitions:

```c
typedef uint32_t size_t;
#define SIZE_MAX 0xffffffffUL
void *malloc(size_t size);
```

Your solution shouldn’t assume compiler support for 64-bit `long long` integer types.

Problem 3 Imagine a modified x86 processor in which saved return address and saved register values (including saved frame pointers) were kept in a separate stack managed by the kernel (called a `shadow stack`). In such a processor, it would not be possible for a buffer overflow on the (regular) stack to overwrite saved return addresses or other saved registers. And no overflows at all would be possible on the shadow stack, since it would contain no buffer or array.

In such a modified x86 processor, would it ever be possible for an attacker to use a buffer overflow on the stack to divert the control flow of the vulnerable program and execute shellcode?

Be concrete. If your answer is “yes,” show a vulnerable program and explain how it can be exploited. If your answer is “no,” explain why no attack is ever possible.
Problem 3 Why would someone want to register domain names such as aazon.com, mic2osoft.com, fjcdn.net, or doubleslick.net?

What would an attacker gain from owning such domains and having servers responding to requests made to them?

Hint: You may wish to refer to an ASCII table.

Problem 4 In class we learned about SYN flooding, a DoS attack that exhausts a server’s “backlog” of half-open TCP connections, and SYN cookies, a cryptographic defense against SYN flooding. With SYN cookies, the server chooses its ISN as a cryptographic function of the client’s ISN, source and destination IPs and ports, and other information from the client SYN packet. The server sends a SYN+ACK with this specially chosen ISN, but does not keep any state; it can check the cookie from the acknowledgment number in a legitimate client’s ACK packet and then establish a connection.

But why are SYN cookies necessary at all? Consider the following alternative: When the server receives a SYN (with ISN $x$), it generates a totally random ISN $y$, sends a SYN+ACK back with ISN $y$, but does not keep any state. (In particular, it does not record either $x$ or $y$.) When the server receives an ACK from a legitimate client, it establishes a connection; it can reconstruct what the client’s ISN $x$ must have been from the sequence number in the ACK packet, and what its own ISN $y$ must have been from the acknowledgement number in the ACK packet.

What is wrong with this approach? Under what conditions will it provide less security than SYN cookies? Be specific.

Problem 5 Suppose you are driving along the highway and see the sign pictured below:
Where will the police actually be waiting? Explain. It may help to explain the threat model you are assuming.

(In case it is not clear from the picture, the sign is immediately before the offramp for exit 107 from I-20 in Louisiana to Camp Road, which is also known as Ouachita Parish Road 25. The next exit from I-20 after exit 107 is not for another four or five miles.)

**Hint:** Why put a sign up at all?