[*The exam text is in English. You should answer the questions either in English or in Dutch. In either language and for all answers you should be concise!*

The exam consists of 5 questions (4 pages). Mark each answer with the question number. The score is indicated per question. The maximum number of points you can get is 126, which corresponds with grade 10.0.

The exam starts at 10.00 hrs and ends at 13.00 hrs. Participation in the exam requires being present for at least 1 hrs.]

Electronic devices are not allowed unless and must be stowed.
Points for each question are indicated.

Every answer requires an explanation. Answers without explanations get no points, even if the answer itself is correct. Incorrect or irrelevant explanations may reduce the number of points awarded even if a correct explanation is also present. Encrypted answers are not allowed.

Question 1 – true or false with explanation

Question 1a (3 points)

A new CPU is developed where the stack grows upwards (towards higher memory addresses); does this solve the problem of stack-based buffer overflows?

Question 1b (3 points)

Symmetric cryptography cannot offer non-repudiation

Question 1c (3 points)

Alice writes a message and sends it to Bob in plaintext. She also sends a signature by encrypting its hash using her private key. Bob uses Alice's public key to verify the signature. Mallory can read and modify data sent between Alice and Bob. Mallory has broken second pre-image resistance for the hash mechanism used. She can use this to get Bob to accept a forged message as if it came from Alice.

Question 1d (3 points)

Any website can be made more secure by setting the "secure" flag on cookies

Question 1e (3 points)

A third-party advertisement in an iframe can read the cookies of the main website

Question 1f (3 points)

Access Control Lists (ACLs) using Role-based Access Control (RBAC) are more secure than ACLs without roles

Question 2 – assembly/shellcode

You found a SUID root program that uses an unchecked `strcpy` to write to a fixed-size buffer on the stack. The program runs on 64-bit x86 Linux. You want to exploit it by placing the following shellcode in its environment and pointing the return address to it.

```
 1 leaq string_addr(%rip), %rdi
 2 movb $0, 0x07(%rdi)
 3 movq %rdi, 0x08(%rdi)
 4 movq $0, 0x10(%rdi)
 5 leaq 0x08(%rdi), %rsi
 6 movq $0, %rdx
 7 movl $59, %eax
 8 syscall
 9 string_addr:
10 .ascii "/bin/shNAAAAAAAABBBBBBBB"
```

Question 2a (5 points)

What is wrong with this shellcode? What will happen if you try to perform the attack with it?

Question 2b (15 points)

How would you fix this shellcode? For every line that causes problems, indicate:

- The line number
- What is wrong
- How you would change the code to fix it

Question 3e (5 points)
The same code is now protected using (only) W⊕X. Can you still exploit it? If yes, how? If no, why not?

Question 3f (5 points)
The same code is now protected using (only) a shadow stack. Can you still exploit it? If yes, how? If no, why not?

Question 4 – web security
You connect to a banking website to perform an online payment. You log in using your username and password and before you can transfer money. The website uses SSL.

Question 4a (10 points)
When your browser establishes its connection to the server, which checks does it perform to authenticate the server? Name at least 5.

Question 4b (10 points)
If the client does not properly authenticate the server, which attack would the system be vulnerable to? Explain how this attack can be used to perform fraudulent transfers.

Question 5 – cryptography

Question 5a (10 points)
Mzld zm zsszbj zfzhmrs sgd Bzdrzq bhogdq zmc dwokzhm gnv hs vnqjr.

Question 5b (10 points)
Erik wants to use the perfect encryption offered by the one-time pad, but the large key size is problematic. To fix this, he randomly generates a 256-bit key using a strong cryptographic pseudo-random number generator and repeats it as many times as needed to create a key for the one-time pad. As an attacker, you obtain a long stream of ciphertext generated this way. What information can you recover and why?

Question 5c (10 points)
What is Cipher Block Chaining (CBC)? Compared to Electronic Codebook (ECB), does it improve confidentiality? Does it improve integrity? In cases where it does, which type of attack does it prevent?

Question 3 – application security

A web server runs the CGI program shown below. This particular server passes the Cookies header on to the CGI program without any processing or validation. Assume that all possible protections (such as address space layout randomization, canaries and non-executable stack) are disabled and that variables are pushed onto the stack in the order in which they are declared with no additional padding.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char **argv) {
  char *cookies = getenv("HTTP_COOKIE");
  char *cookies_end = cookies + strlen(cookies);
  char *nameptr;
  char name[64] = "";

  nameptr = name;
  while (cookies < cookies_end) {
    /* find where current cookie ends */
    char *cookie_end = strchr(cookies, ';');
    if (cookie_end == NULL) cookie_end = cookies_end;

    /* add "name" cookie to name */
    if (strncmp(cookies, "name=", 5) == 0) {
      size_t len = cookie_end - cookies - 5;
      memcpy(nameptr, cookies + 5, len);
      nameptr += len;
    }

    /* move on to next */
    cookies = cookie_end + 1;
    while (cookies < cookies_end && *cookies == ' ') cookies++;
  }
  *nameptr = 0;

  printf("Content-Type: text/html\n");
  printf("\n");
  printf("<html><body>welcome %s!</body></html>\n", name);
  return 0;
}
```

Question 3a (8 points)

Show the stack layout, including start address of each entry, right before the call to `memcpy`. Assume the return address of `main` is at `0x7fffffffe800`.

Question 3b (10 points)

Explain why this code is vulnerable and how you can exploit it to launch a shell. You should describe your inputs in enough detail to make it work (and not crash the program) but there is no need to give the exploit code or shellcode.

Question 3c (5 points)

The same code is now protected using (only) stack canaries. Can you still exploit it? If yes, how? If no, why not?

Question 3d (5 points)

The same code is now protected using (only) ASLR. Can you still exploit it? If yes, how? If no, why not?