Closed book. Closed notes. No electronic device.

1.

For each description below, give **one** term that *best* describes it. In many cases, but not all, the term will be in the table at left. In each case, give only **one** answer of **at most** 4 words (otherwise you get zero).

ASLR
Buffer overflow
Canary
`chroot()`
CodeRed
CSRF
Clickjacking
Cookie
Cookie elevation
Document virus
Effective uid
Flash cookie
Kerckhoff's principle
Least privilege
Metamorphic
Mitigation
Polymorphic
Ransomware
Real uid
Rootkit
Same origin policy
`seccomp`
Session elevation
Setuid bit
Session cookie
`seteuid()`
`setuid()`
URL's scope
Virus
Web storage
Worm
X-Frame-Options
XSS
Stored XSS
Reflected XSS

1. This determines whether the effective uid of a process changes when it starts executing a file.
2. This can change the real uid of a process.
3. This infects a running program.
4. This infects a stored program.
5. This detects stack smashing.
6. This kind of virus alters its code at each generation.
7. This kind of virus alters its memory image at each generation but its code does not change.
8. Virus contained in a macro of a Word file.
9. Virus contained in the autoplay file of a USB stick.
10. Information that a website gets concerning a client's platform.
11. State that is created at browsers by http headers.
12. Website-specific state at browsers that can be sent to websites only via scripts.
13. State that can be created by a browser and shared across other browsers running on the same OS.
14. This determines which cookies to include in a http request.
15. A website increases the privileges of an existing cookie.
16. A Linux system call that prevents the executing process from opening any more files.
17. This is a way to have framebusting without resorting to javascript.
18. A browser has pages `a.com` and `b.com` open. Page `a.com` sends a request with an attack script to `b.com`; the latter sends a response containing that script; the browser executes the script.
19. This is a way to limit the filesystem available to a process to a subtree of the original filesystem.

2

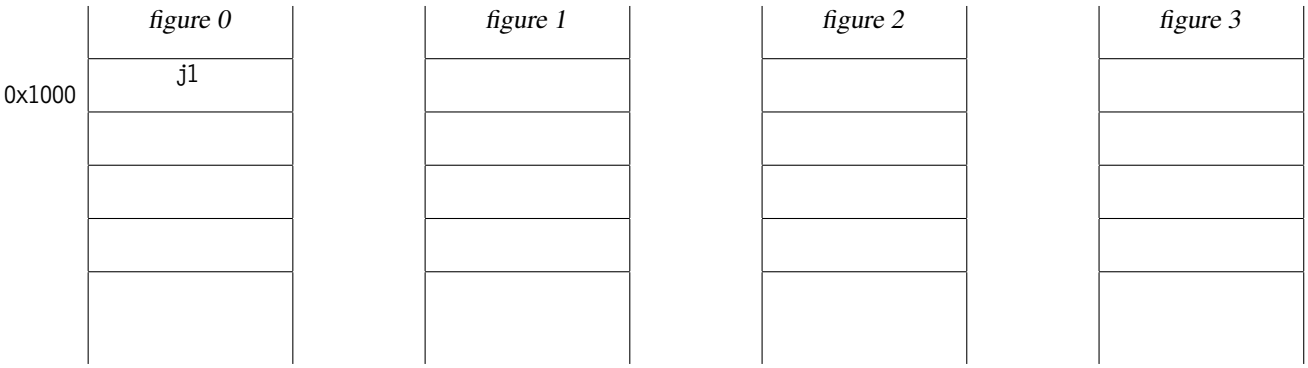
```
int f(int arg) {
    int j3;
    char buf[16];
    FILE *badfile;
s1: badfile = fopen("badfile", "r");
    fread(buf, sizeof(char), 40, badfile);
s2: return arg + 1;
}

int g(int arg) { ... }

int main(int argc, char **argv) {
    int j1;
    int j2[4];
    j1 = 123;
    return f(j1);
}
```

Assume the Ubuntu environment of project 1.

This program is compiled without Stack Guard and executed such that variable `j1` is always at address `0x000001000`.

Below, "draw the stack layout" means indicate, in the referred figure, the contents of the stack from address `0x1000` to the top of stack, and give the addresses (in **hex**) of the contents at the side. Use "`&g`" to indicate the address of function `g()`. The stack grows **downward**.

Below, "`f()` returns to `g()`" means that when control returns from `f()`, it starts executing `g()` with any argument. And "`f()` returns to `g(4)`" means that when control returns from `f()`, it starts executing `g()` with an argument of 4.



2a. In *figure 0*, draw the stack layout when control comes to `s1` (i.e., "`badfile = fopen(..)`" is to be executed next).

2b Assume `badfile` is such that `f()` returns to `g()`. In *figure 1*, draw the stack layout when control comes to `s2` (i.e., "`return arg+1`" is to be executed next). (Use "`&g`" to denote the address of function `g()`.)

2c Assume `badfile` is such that `f()` returns to `g(4)`. In *figure 2*, draw the stack layout when control comes to `s2`.

2d Does your answer for part 2a change if Address Randomization is turned on. Explain briefly.

2e Does your answer for part 2a change if the stack is made Non-Executable. Explain briefly.

2f Now assume the program is compiled with Stack Guard. In *figure 3*, draw the stack layout when control comes to `s1`. Can file `badfile` be initialized so that `func()` returns to `g()`. Explain briefly.

3 Here are two files owned by root:

- `/passwd.txt`: text file that contains user passwords. Root has read-write-execute access. All other users have no access.
- `/chpwd`: executable file that users can run to change their passwords. Root has read-write-execute access. All other users have write-execute access. The setuid bit is set (so it is a set-root-uid file).

Does this configuration allow an ordinary (i.e., non-root) user to delete `passwd.txt`?
If no, explain briefly.
If yes, briefly give the steps of the attack.

4 Website `a.com` has an SQL table `Users` with a row for every user and columns `Name`, `Pwd`, `Age` and others.

- A user changes its password with the GET request
  `http://a.com/chpwd.php?a1=<name>&a2=<opwd>&a3=<npwd>`.
  The server handles this with
  `chpwd.php:`
  ```
  $db->sql_query("UPDATE Users SET Pwd='$npwd' WHERE (Name='$name' AND Pwd='$opwd');");
  ```
- A user gets the names of all users with the GET request
  `http://a.com/getusers.php`
  The server handles this with
  `getusers.php:`
  ```
  $db->sql_query("SELECT User FROM Users WHERE 1=1;");
  ```
- Among the users are `Ted` and `Bob` (these are their `Name` entries). `Ted` does not know `Bob`'s `Pwd` value.

3a. Give the path of a GET request that `Ted` can issue in order to change `Bob`'s password to `fqr123`.

3b. Rewrite the `chpwd` code to use the *prepare* construct.

5 This problem concerns a browser `c1`, website `s1`, and attacker website `s2`.

- `c1` clicks `http://s1/p1.html`. In the response, `s1` sets a cookie for domain `s1`.
- Then `c1` clicks `http://s2/p2.html`.
- Then `p1.html` regularly issues POST requests to `s1`. Each POST request contains the cookie value in its data. The server treats a request as valid iff the cookie value (in the request header) matches the value in the data.

For each of the following cases, answer whether `p2.html` can send a POST request to `s1` that the latter treats as valid. Write "YES" if it can, and "NO" if it cannot. (Below, "unguessable" is equivalent to "randomly generated".)

s1-cookie name	**s1-cookie value**	**your answer**
guessable	guessable	
guessable	unguessable	
unguessable	guessable	
unguessable	unguessable	

6 When `deposit()` is called with `user` pointing to a valid `User` instance, it should update the `user`'s `balance` or return `0` upon error. Does the above function achieve that. If not, fix the function.

```
struct User {
   unsigned int balance;
};

unsigned int deposit(struct User *user, unsigned int amount) {
   user->balance += amount;
   return user->balance;
}
```

7 The following function is in a program owned by root and executable by any user. `ustr` points to a string supplied by the user. `secret` points to a string that is stored within the program and should not be disclosed to the user. The function prints a message indicating the result of `strcmp(ustr,secret)`. Can the function expose the `secret` string. If so, fix the function.

```
int f(char* ustr, char* secret) {
   printf(ustr);
   if (strcmp(ustr, secret) <= 0)
      printf(" <= secret");
   else
      printf(" > secret");
}
```