Question 1: An Authentication Program

Checkpoint 0. Give the login command needed to login as one of the users in the database.

The command form is `./Login <Username> <Password>`. If you examine the database you can find the user names along with hashed passwords. You can try to break one of the passwords with a dictionary attack, or make an educated guess such as:

```
./Login user user
```

Checkpoint 1. What does the output of `findbugs` tell you?

FindBugs tell you that there is a possible SQL injection (CWE-89) because user input is used directly to create SQL statement by string concatenation. FindBugs also tells you that the statement is not closed after use.

Checkpoint 2. Which part of the authentication program is vulnerable to SQL injection and how can an attacker exploit it?

In the method `doPrivilegedAction`, the `user` parameter is used directly for the creation of SQL statement by string concatenation without any escaping or checking process. An attacker can add arbitary SQL code to corrupt the database or affect the retrieved result.

Checkpoint 3. Why isn't the removal of quotes and semicolons through `sed` adequate to protect against SQL injection?

Firstly it doesn't remove all SQL special character like comments or wildcard characters. More importantly there is an `echo -e` after the sed command which will interpret all '\' escaped hex characters which may expand to quote and semicolons characters as their ASCII values.

Checkpoint 4. Give your patch to repair the `Login` and `Login.java` programs and explain how it works.

Use absolute path for `sed` command. Avoid `echo -e` command after `sed` operation Use a prepared statement in Java for database query preperation

Checkpoint 5. Examine the exploits and explain how they work. What design flaws or vulnerabilities are abused by the exploits?

Both of them try to shortern the SQL query. One of them provides the hex value representation of quote and semicolons which will be interpreted by the echo command. This is the problem of "time of check to time of use". The other one lets the Linux system search for the `sed` binary in a self-defined directory, which allows a malicious sed to be executed. This is abuse of the `PATH` variable as linux will search for unknown executable binary in the order of directory listing provided by PATH variable.

Checkpoint 6. Did your patch stop the exploits? If not explain why and provide an updated patch that does.

Question 2: Another SQL injection

Checkpoint 0. What are the users already in the database and what attacks against them can you imagine?

The users are `admin` and `user`. A second order SQL injection attack is possible. This allows an attacker to change the password for any user illegally.

Checkpoint 1. When you run the `findbugs` program on the class file it shows there is an SQL injection problem despite the use of a prepared statement. Why?

Even though the username has been saved to the database with a prepared statement, we don't check it for SQL metacharacters. Consequently whilst we cannot do any exploits while logging in or signup, we can do some tricks while changing the password.

Checkpoint 2. Fix the code and provide a patch for `Login.java`.

Avoid naked string concatenation on creating sql query. Use prepared statement instead. Malicious user input may not have immeidate effect, it may work if it is reused in some later operation. It is called as second order SQL injection. So it is better to check for and deny SQL metacharacters before storing into the db.

Checkpoint 3. Describe how the exploit works.

It is a second order SQL injection. If you create a user with a silly username like admin';– You can perform bad operation later to mess up the query logic and overwrite an admin password.

Checkpoint 4. Verify your patch stops the exploit. If not, make a new patch that does.

Question 3: Linkers

Checkpoint 1. What program is used to interpret the ELF file `Vulnerable`?

/lib/ld-linux.so.2

Checkpoint 2. What is an untrusted search path? How it related to dynamically linking of library?

The binary tries search for some resources or libraries from an externally provided location. This is named as untrusted search path. Dynamic linking of libraries make the binary to find some of the resources or needed library classes at run time, which provide a chance for an attacker to direct the binary to a malicious or modified resources or library classes using untrusted search path technique which the binary have no control or maybe not aware of the action.

Checkpoint 3. How could the `Vulnerable` program be fixed to avoid the search path exploit?

Use static linking of libraries. There is more suggestions on CWE-426 Untrusted Search Path documentation.

Checkpoint 4. Describe what each of the provided exploits does, how likely you consider it to be an achievable exploit and under what circumstances.

One cracks the password… given the complexity of the password it is unlikely to be cracked quickly, however this is still a threat as SHA1 isn't the greatest hashing algorithm for passwords. Also, attacker may look for carlessly stored password or hash by reverse engineering of the binary itself.

One uses preloading… very likely given the implementation, however if the program ran as a setuid program *most* modern linkers would ignore it.

The last modifies the binary and removes the conditional jump based on the output of the `memcmp`. Extremely likely (this was a common way of cracking software).

Checkpoint 5. How could you prevent each of the attacks?

Password Cracking Defend by obfuscating the password, and considering a different authentication scheme.

Preloading Defend by statically linking.

Binary Patching Near impossible to stop. Solutions can include using specialist hardware (i.e. ROM with some hardware protection) to store and run the program, or using a cryptographic packer to provide obfuscation.

Checkpoint 6. (optional) Modify `exploit-ld.sh` so it attacks the `SHA1` call. Similar to exploit-ld.sh Provide a modified version of SHA1 call which always copy the needed hash value to the dest address provided in the argument. Use memcpy instead of strcpy or strncpy to avoid problems caused by zero byte in the hash.