# Secure Sw engineering - SE ZG566 regular

## Question 1: Supply Chain Attack

1. Does the pesticide paradox apply to security testing?
2. Briefly explain the Java security architecture.
3. Do you think NoSQL databases are more secure than SQL databases?
4. What are relative strengths of symmetric and asymmetric encryption?
5. What are the challenges in finding a good dataset to develop a robust IDS?
6. How will you estimate the cost of a breach?

## Answer

1. Pesticide Paradox: Yes, it does When a small number of modules contain most of the bugs detected or show the most operational failure similarly when same tests are repeated over and over again until they can find no more bugs

2. JAVA security architecture: It is mostly focused on protecting users from hostile programs downloaded from untrusted sources across a network. To accomplish this JAVA uses sandbox in which JAVA programs run

3. NoSQL security: No, I don't think NoSQL is more secure than SQL. Infact, NoSQL have very few inbuilt security features in order to allow faster data access.

4. Strengths of symmetric and asymmetric encryption: Symmetric encryption is significantly faster than the latter. Asymmetric encryption though slow need no private key to be shared making it more secure. A secret is inverserally proportinal to number of people knowing it

5. Challenges in finding good dataset for IDS: Estimating the difficulty typically involves comparing state of the art IDS. False alarm rate and low detection rate can be an issue if we don't find a good dataset.

6. Asessing the type of breach: In assessing the breach we need to consider type or types of personal information involved in breach, the circumstance of the breach, including cause and exent, the nature of harm to affected individuals and if this harm can be removed through remedial action.

# Question 2

List features of a penetration testing tool with help of an example. Do you recommend a development team member to carry out penetration testing?

## Answer

1. Detailed and comprehensive reports
A good penetration testing software must be able to give detailed and comprehensive reports. Penetration testing does not just end at finding vulnerabilities in a network. The operator or administrator must be able to understand the problems in the network. Without this knowledge, it would be challenging to plan the following action. A testing report should describe, give evidence for and assess the risk, and recommend the solution to any vulnerability found.

2. Built-in vulnerability scanner
A vulnerability scanner comes hand in with most commercial penetration testing tools. The purpose of vulnerability scanning is to find any hardware or software lapse that may prove a route of attack on the system later. Vulnerability scanners carry out their scanning based on a published database of Common Vulnerabilities and Exposures (CVE), making regular updates very important. Scanning also includes automated scans that one can set to run across specific applications.

3. Multi-system operability
An indispensable feature of good penetration testing software is its ability to be used on different devices. Most penetration software is Linux operating systems compatible, with some already pre-installed on the OS. However, other devices running on Windows operating systems, macOS, and Android mobile phones also need penetration testing tools to check for vulnerabilities. This has made testing software that is compatible with several devices to be in high demand.

4. Password cracking capabilities
Passwords form one of the weakest links in any organization or computer network. Individuals often use the most basic combination of characters to safeguard access to vital information. This is why penetration tests often include an assessment of password strength. As such, penetration testing software must be able to crack passwords. They use a combination of features such as brute force attacks, cryptanalysis attacks, and dictionary attacks to assess password strength.

# Question 3

How does SQUARE augment conventional requirements engineering?

## Answer

SQUARE stands for Security Quality Requirements Engineering. It is a requirements engineering process developed for eliciting and documenting security requirements. Since security requirements are often not given the focus that they deserve and since trying to incorporate security requirements later in the software development lifecycle costs more than planning for them upfront, the SEI developed a nine-step process to ensure that quality security requirements can be gathered, categorized, prioritized, and validated early on in the software development lifecycle. These nine steps are:

1.Agree on definitions
2.Identify security goals
3.Develop artifacts
4.Perform risk assessment
5.Select elicitation techniques
6.Elicit security requirements
7.Categorize requirements
8.Prioritize requirements
9.Inspect requirements

# Question 4

List significant components of the attack surface of a Web Application

## Answer

Attack surface of web app

1. Open ports on outward facing web and other servers and codes listening on these ports
2. services available on inside of firewall
3. codes that process incoming data, email, XML, office documents, industry specific custom data exchange format
4. Interfaces , SQL , web forms
5. an employee with access to sensitive information , vulnerable to social engineering attack

# Question 5

```
#include <stdio.h>
#include <string.h>
void hello_func(char *name)
{
char buf[20];
strcpy(buf, name);
```

```
printf("Hello %s\n", buf);
}
int main(int argc, char *argv[])
{
hello_func(argv[1]);
return 0;
}
```

## Answer

Concerns with The above program are:

1) If the program is not passed an argument the memory won't be allocated for argv[1] hence it can write to an unknown memory location and corrupt the program
2) char* name has an unknown length and it might not fit in buf and overwrite other memory
3) Printing an memory location without knowing that the string is null terminated or not may cause the program to crash or print some other data of the memory.

All these writes on memory can be used to inject malicious code into the memory and can cause harm to the system.

# Question 6

```
int concatStrs(char *bufl, char *buf2, unsigned len1, unsigned len2) {
char tempBuf[256];
if ((len1+len2) > 256)
return FAILURE;
memcpy(tempBuf, bufl, len1); memcpy(tempBuf+ len1, buf2, len2);
processStr(tempBuf);
return SUCCESS;
}
```

## Answer

Increasing source size in memcpy() can cause undefined behaviour.
memcpy() doesn’t check for overflow or \0
memcpy() leads to problems when the source and destination addresses overlap.
Software attacks Integer overflow
Software attacks The problem The previous function is not safe, because integer types have finite precision. So the previous function could have an integer overflow, which in turn (in this example) can lead to a buffer overflow
Software attacks Integer overflow Integer overflows occur when a larger integer is needed to accurately represent the results of an arithmetic operation.

Software attacks Integer overflow in new[] But how could you get tricked into an overflow situation? The most common way of doing this is by reading the value out of a file or some other storage location. For example, if your code is parsing a file that has a section whose format is "length followed by data“ See GIF, JPG, BMP vulnerabilities

# Question 7

You are the Security expert for an ecommerce provider. They maintain their data in an RDBMS. They plan to offer self-service feature for product queries. Customers can search for products using a variety of criteria. The software assembles appropriate SQL query, executes it, and shows results to customers. Prepare a set of guidelines for developers building the functionality based on your knowledge of software vulnerabilities. [5 marks]

1. Authentication and Authorization - Any transaction which requries updated of a the persisted data must be authenticated and authorized and must not be exposed to anonymized users.
2. The APIs must not be accessible only on private web where in the web server is able to access the APIs , not by a public domain.
3. The APis must  restrict the users to specific resource usage
4. The forms must include validation to avoid SQL injection attack
5. The APis must not directly evaluate input from front end , it should rather validate the same for malicious code

# Question 8

You are a product development organization which needs to regularly release patches for your customers distributed across the globe. List (and justify) the steps taken by you to ensure that your customers receive genuine software.

## Answer

1. Securing data in transit , though termed data , software patches are programms but treated as data during delievry to distributed locations. In-transit security is enforced by encrypting the patch with encryption mechanism such as symmetric or asymmetric keys as suitable.
2. Hashing the streamed software patch such that it's integrity can be validated as receiving end
3. Digitally signing the patch package to ensure authentication and authorization is possible at receiving end
4. Setting up validation mechanism at distributed sites to authenticate incoming packages.
5. Establishing direct private conencts , example SSH between deployment and end site to avoid sniffing

# Question 9

You are the CISO (Chief Information Security Officer) of an organization. List the events (in sofMare environment) that you would like to be alerted promptly. Provide a brief justification for each event

## Answer

1. Securing data in transit , though termed data , software patches are programms but treated as data during delievry to distributed locations. In-transit security is enforced by encrypting the patch with encryption mechanism such as symmetric or asymmetric keys as suitable.
2. Hashing the streamed software patch such that it's integrity can be validated as receiving end
3. Digitally signing the patch package to ensure authentication and authorization is possible at receiving end
4. Setting up validation mechanism at distributed sites to authenticate incoming packages.
5. Establishing direct private conencts , example SSH between deployment and end site to avoid sniffing