<u>Assignment</u>

Name: K Ravi Kumar Reddy
BITS ID: 2020MT13010
Course ID: SSZG570
Course: Cloud, IoT & Enterprise Security

Question 1:

Slashdotted: Also known as Slashdot effect or slash-dotting, this is the phenomenon of linking a popular website to a smaller site, causing the smaller site to slow down or even temporarily close due to the increased traffic.
This has the same effect as a denial-of-service attach. The name originates from the huge influx of web traffic which would result from the technology news site Slashdot linking to websites.
One of the solutions for a small site getting Slashdotted is employ systems that automatically mirror any Slashdot-linked pages to ensure that the content remains available even if the original site becomes unresponsive. Sites in process of getting Slashdotted may be able to mitigate the effect by temporarily redirecting requests for the targeted pages to one of these mirrors.

Flash Crowd: A flash crowd is a more generic term without using any specific name that describes a network phenomenon where a network or host suddenly receives much traffic. This is sometimes due to the appearance of a website on a blog or news column.
This term is commonly used in internet to describe exponential spikes in website or server usage when it passes a certain threshold of popular interest.

Relation with DoS attacks:

The major similarity between Slashdotting, Flash Crows and Denial of Surface (DoS) is the network getting overloaded and the consequences of the phenomena. Both result in making the website/resource unavailable to its intended users. While slashdotting is a legitimate phenomenon to occur when a popular site refers a smaller site and the latter receives an enormous amount of traffic, DoS is a cyber-attack in which a perpetrator intentionally seeks to make a machine or resource unavailable.

On the other hand, there are historic occurrences of social media accounts of celebrities getting hacked and being used to comment on smaller sites thereby resulting a flash crowd on the smaller sites which results in their unavailability.

Steps to be taken when DoS attack is detected:

DoS attack targets the availability of the system that offers services. This is achieved by exhaustingly consuming resources at the victim so that the offered services become unavailable to legitimate entities. One can follow following steps when a DoS attack is detected to mitigate its adverse consequences:

a. Graceful Degradation: Consider how you will best serve some of the needs of some of your users during an attack.
    - Discuss with service providers about their ability to immediately implement any responsive actions.
    - Transfer all online services to cloud-based hosting hosted by major cloud service providers with high bandwidth and CDNs that cache non-dynamic websites to obtain redundancy.
    - If using a content delivery network, avoid disclosing the IP address of the origin web server and use a firewall to ensure that only the CDN can access the web server
    - Use a DoS mitigation service for the duration of the attacks. Disable functionality and remove content from online services that enable the current DoS attack to be effective deliberately.

   - Prioritize access based on its source. For example, limit access to only UK IP addresses if the source attack IPs are UK registered.
   - Disable dynamically generate content such as site search or customer specific product recommendations and so on.
   - Only successfully authenticated users can be provided with the dynamically generated content to ensure their experience is not affected by the attack

b. Dealing with changing tactics and repeated attacks:
   - DoS attacks generally occur in waves to overwhelm the system in different ways as the attackers learns about various protections established.
   - As a defense strategy, IP masking and extreme security measures should be employed with an efficient network load balancer and IPS system.

c. Retaining administrative access:
   - Management access is mostly likely to be disabled when the attack occurs.
   - An alternative access should be installed via different subnet and constrain this access to an allow list of trusted locations.
   - An important point to remember while installing an alternative management access is to not rely on the same public DNS Zones used earlier or the ones that likely to be targeted.

d. Restoring Services:
   - Taking regular backups of critical files, configuration files of servers, routers and firewalls on regular should be a mandatory practice.
   - These backups can be used to restore services as necessary.

Measures to trace the source of packets used in DoS attack:
If the attack is a simple DoS attack (coming from only one IP address), then it is *easier* to trace. A network trace using Wireshark or tcpdump will show enormous number of packets from one IP address. Blocking this IP alone would halt the attack immediately. Most firewalls block commonly used IPs in DoS attacks as the original locations of them are generally disguised or in through botnets which are generally not from one source.

However, if the situation is complex such as a DDoS attack, there will be requests from several addresses. While simple network traces can give the IPs, tracing systems will need to be employed as the number if IPs increase. There are two categories of tracing systems used in tracing the DoS attack – proactive and reactive tracing systems.

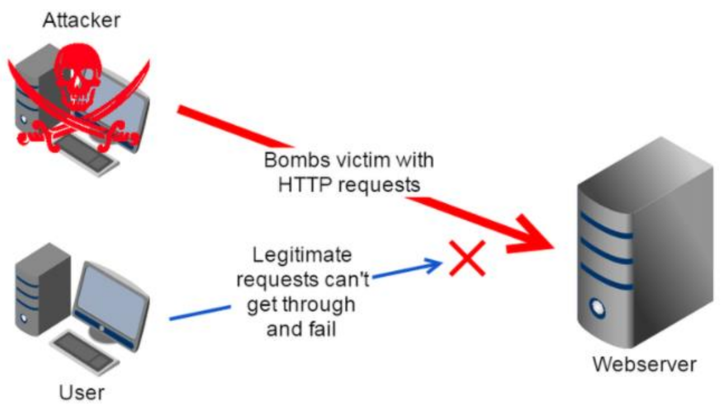
Proactive Tracing Systems:
In the proactive tracing, the system records packet information during packet transmission. Once the attack occurs in the network, investigators can use this information to track routing paths and identify a source of attack.

- *Packet Marking Scheme:* In this scheme, the information of the routers is stored in the packets that it passes. So, the recipient of the marked packet is able to use this router information to route the packet's path to its source.
- *ICMP Traceback Scheme:* In this technique, each router that suspicious packets passthrough generates an ICMP trace back message or iTrace. Typically, the iTrace message consists of the hop information, and a timestamp. This information will be used as a path recovery to identify the routing path back to an originated source. Similarly, to a packet marking scheme, the disadvantages of this scheme are a requirement of a large number of packets to reconstruct attacking paths, as well as a modification of routers in which they can support a feature of routing information addition. Also, the ability to prevent major DoS and DDoS attacks is bad because these techniques cannot deal with a large number of reflectors.
- *Hash-based IP Traceback Scheme:* The router, called Data Generation Agents(DGAs), records packet information from packets passing through it. The other components in this technique consist of:
   - SPIE Collection and Reduction Agents (SCARs) which are used for query necessary information from connected DGAs

- SPIE traceback manager (STM) which is used to communicate to the victims IDS through a central managing unit. This scheme reduces the size of storage space to store path information by using hash functions.

Reactive tracing systems:
Different from proactive tracing, reactive tracing starts to record packet information and traceback to an originated IP address after the attacks have been detected. The challenge of this technique is to develop methods that be able to detect and match the attack effectively.

- *Hop-by-Hop Tracing:* This method is suitable for tracing denial-of-service attacks because the nature of DoS traffic is large and continuous packet flows. Moreover, the source of attack is mostly spoofed IP addresses. In this technique, a tracing program which is installed into the router located closest to the attacker, is used for monitoring incoming packets. If attackers use spoofed IP address to launch attack, these packets will be stored into the routers for monitoring. This procedure is repeated to the adjacency routers until the program can identify the attacker's originated IP address. On the other hand, the limitation of this scheme occurs when it deals with DoS/DDoS attacks in which the packets come from multiple sources and each source generates only a small number of packets. So, the information recorded in some routers might not enough for a program to track attackers back to their original source. Similarly, if the attacker stops their action before the trace is finished, the examination will be failure.
- *Overlay Network:* Generally, the standard function of routers only examines a packet header and chooses the best path to forward packets to intended destination. Stone has proposed a technique to add a traceability feature to router in order to recover attacking path back to attackers. This technique can be accomplished by adding a device called tracking router to monitor all traffic passing through this network. In this scheme, even it does not require any modification on the user's router to run a tracing process, the drawback is in many requirements in ISP involving in order to identify the attack source IP address. Moreover, the high processing overhead over each packet is another drawback of this scheme


Attacker
Bombs victim with HTTP requests
Legitimate requests can't get through and fail
Webserver
User

Question 2:

Given conditions are:
Rajesh wants to send a message X (assumed to be an integer mod N) to Ramesh, he computes the value E(x) ≡ XE (mod N) and sends this to Ramesh. At the reception side of the message, the value y = E(x), Ramesh computes D(y) ≡ yd (mod N); this will be equal to the original message x. where E and D are functions to encode and decode.

a) **RSA Encryption Algorithm** can be designed from the above conditions.

   This is because RSA algorithm uses asymmetric encryption which encompasses of two different keys public key and private key wherein public key is shared and private key is not. In the above conditions also, Rajesh and Ramesh are using two different keys one for encrypting and one for decrypting. Further, RSA is one of the safest encryption algorithms and has been used by various organizations to this day. This is because this algorithm is very hard to crack since it involves factorization of very large prime numbers.

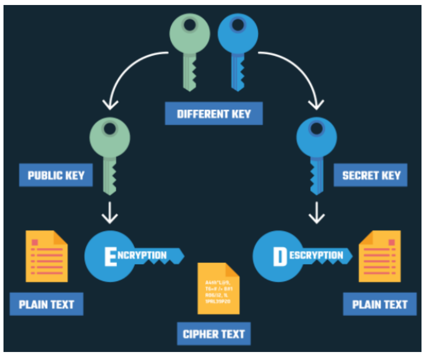
b) In the given conditions:
   Prime Numbers: p, q (512 bits)
   N = pq
   e = Number prime to (p-1)(q-1)
   Public Key = (N,e)
   Let M = (p-1)(q-1)

   Private Key can be calculated as shown below:
   ed = 1 (mod M)
   d = 1(mod M)/e

c) Three major components of the RSA algorithm are exponentiation, inversion and modular operation. Time complexity of the algorithm heavily depends on the complexity of the sub modules used. We can take the liberty to perform modular addition in cryptography in $O(logn)$ where n is the number of digits in any number in $\mathbb{Z}$.

   Now modular multiplication using squaring and multiply technique to get $m^e modN$ can be shown as multiplying the digits of m log e times. Considering the complexity of multiplication $O(\{logn\}^2)$ i.e., repeated addition of two number of logn bits each, the complexity of the modular exponentiation is about $O(\{logn\}^3)$
   Using Euclidean extended GCD from (Extended Euclidean algorithm), inverse of a number can be calculated in $O(\{logn\}^2)$

   Thus, overall time **complexity of the key generation algorithm will be $O(N^2)$ encryption** and **decryption algorithm using squaring and multiply technique will be $O(N^3)$**: For N digit number space.



Question 3:

Attack Surface Analysis: It is about mapping out what parts of a system need to be reviewed and tested for security vulnerabilities. The point of Attack Surface analysis is to understand risk areas in an application, to make developers and security specialists aware of what parts of the application are open to attack, to find ways of minimizing this, and to notice when and how the Attack Surface changes and what this means from a risk perspective.

Attack Surface analysis is usually done by security architects and pen testers. But developers should understand and monitor the attack surface as they design and build and change a system. Attack Surface Analysis helps you to:

- Identify what functions and what parts of the system you need to review/test for security vulnerabilities.
- Identify high risk areas of code that require defense-in-depth protection – what parts of the system that you need to defend
- Identify when you have changed the attack surface and need to do threat assessment.

The Attack Surface of an application is:
The sum of all paths for data/commands into and out of the application, and the code that protects these paths (including resource connection and authentication, authorization, activity logging, data validation and encoding) all valuable data used in the application, including secrets and keys, intellectual property, critical business data, personal data and PII, and the code that protects these data (including encryption and checksums, access auditing, and data integrity and operational security controls). You overlay this model with the different types of users - roles, privilege levels – that can access the system (whether authorized or not). Complexity increases with the number of different types of users. But it is important to focus especially on the two extremes: unauthenticated, anonymous users and highly privileged admin users (Ex. Database administrators, system administrators).

Entry and Exit points for an Enterprise like Amazon:

1. User Interface Forms and Fields: Every field of type 'input' in the UI of every website or portal in hosted in the enterprise is susceptible to XSS attacks, Bruteforce attacks and so on.
2. HTTP headers and cookies are a major entry point
3. APIs
4. Files
5. Databases
6. Other local storage
7. Email or other kinds of messages
8. Run-time arguments
9. Login/authentication entry points
10. Admin interfaces
11. Inquiries and search functions
12. Data entry (CRUD) forms
13. Business workfloes
14. Transactional interfaces/APIs
15. Operational command and monitoring interfaces/APIs
16. Interfaces with other application/systems

In addition to these, valuable data (ex. Confidential, sensitive and regulated) in the application, by interviewing developers and users of the system and again by reviewing the source code.

1. **Different types of users with different roles and privileges can access the system:** Role-based access control (RBAC), also known as role-based security, is a mechanism that restricts system access. It involves setting permissions and privileges to enable access to authorized users. Most large organizations use role-based access control to provide their employees with varying levels of

access based on their roles and responsibilities. This protects sensitive data and ensures employees can only access information and perform actions they need to do their jobs.

An organization assigns a role-based access control role to every employee; the role determines which permissions the system grants to the user. For example, you can designate whether a user is an administrator, a specialist, or an end-user, and limit access to specific resources or tasks. An organization may let some individuals create or modify files while providing others with viewing permission only.

One role-based access control example is a set of permissions that allow users to read, edit, or delete articles in a writing application. There are two roles, a Writer and a Reader, and their respective permission levels are presented in this truth table. Using this table, you can assign permissions to each user.

2. **Complexity increases with the number and types of users:** While countermeasures might need to be adjusted to accommodate non-traditional schedules (e.g., the practice of limiting users to acceptable log-in times and locations), a system with telecommuters, frequent travelers, and other remote access users can still be secure. Doing so may require policymakers to think more creatively, but each security guideline needs to be customized to meet the organization's needs anyway.

   While most system users are probably trustworthy, it doesn't mean that they're above having occasional computing accidents. After all, most system problems are the result of human mistake. By instituting security procedures, the organization protects not only the system and its information, but also each user who could at some point unintentionally damage a valued file. By knowing that "their" information is maintained in a secure fashion, employees will feel more comfortable and confident about their computing activities.

3. **Need to check for unauthorized, anonymous users and highly privileged admin users:** Cyber criminals are constantly improving their strategies to gain access and wreak havoc. Privileged user accounts, and the credentials that secure them, remain a ripe target. Most attackers take a methodical, multi-step approach to gain access to your most critical systems and data. They often start by taking over user accounts which are using default or common passwords. This is just the foot in the door they need, since their real goal is to take over privileged accounts and escalate their access to applications, data, and key administrative functions. Understand the tactics used by cyber criminals to compromise privileged accounts. These include compromising a local account, capturing a privileged account, performing patient and stealthy recognizance and learning about the normal routines of IT teams, impersonating employees, establishing ongoing access, and causing harm—both in the short-term and over the long haul.

4-8. **Group type of attack point into buckets based on risk purpose, implementation, design and technology, count the number of attack points for each type, finding out various endpoints and assessing their risks.**

Based on risk purpose:

High Risk Attack Points:

- *The Network connection between shopper and website's server:* the attacker can sometimes perform a Man in The Middle (MITM) attack where they sniff traffic on the connection between client and server and record data such as credit card information in transit. Secure Sockets Layer (SSL) encryption has generally solved the problem of credit card numbers being intercepted in transit between the consumer and the merchant but weaknesses in session negotiation and SSL transport protocols are documented and do exist and can be exploited by a sufficiently motivated and resourced attacker.
- *The website's server:* Most vulnerabilities in the e-commerce process relate to the website that hosts the e-commerce site. This is because if compromised the attacker doesn't just get access to a single user's details (as they would targeting a single user

or sniffing their network traffic) but potentially the entire data set of the store, including up to hundreds of thousands of identities and credit card numbers for the largest vendors.

- *Third party software vendors:* attackers' area also able to occasionally exploit security weaknesses in software vendors, such as shopping cart providers. One specific example might be to insert malware such as malicious JavaScript in shopping card code that ships to hundreds of vendors. An unwitting vendor then serves this JavaScript to customers believing it to be legitimate functional code, and all their client's find their credit card details sent without their knowledge to an attacker's server when they enter them on the online store to perform payment.

Low Risk Attack Points:

- *The client (shopper):* the attacker can use methods such as social engineering to gain access to the customer credit card details. One such example is to host a domain that is superficially similar to a reputable supplier, but with a typo or similar Unicode character replacement in the domain name and which masquerades as the trusted domain. If the attacker can get a customer to visit this site via a link, and to submit their credit card details or password believing it is the real site, information theft is possible.
- *The client (shopper)'s computer:* the attacker can also target shoppers' computers using attacks including malware, installed as a result of phishing or social engineering attacks. The malware can include malicious functions such as keyloggers that record passwords the user types and sends them over the web to an attacker's system for capture.

Based on Implementation & Design:

- *Reverse-Engineering:* Reverse-engineering can be seen as an implementation-related attack in the context of embedded systems given access to embedded program code or an implementation in silicon on an IC, an adversary attempts to reconstruct for instance confidential, vendor-specific algorithms. In certain cases, reverse-engineering can serve as a preparatory step for side-channel attacks or fault injection to simplify the profiling of the system or to be able to communicate with the device at all (in the case of proprietary protocols).

- *Logical Attacks:* focus on flaws on the level of transmission and communication protocols. For example, an adversary may eavesdrop on unprotected data exchanges or act as a Man-In-The-Middle (MITM) between legitimate communication partners. Finally, embedded software and hardware implementations can, like every complex system, contain bugs that allow an attacker to compromise the security by, e.g., sending unexpected input data to the system.

- *Fault Injection:* Several new ways for attacking the implementation rather than the underlying mathematical theory open up. For example, an adversary may disturb the computations on a device and subsequently evaluate the "faulty" results. This method termed Fault Injection (FI) and has since then been adapted to most cryptographic algorithms, including otherwise secure primitives like AES, 3DES, RSA, and Elliptic Curve Cryptography (ECC).

- *Side-Channel Analysis:* In contrast to the active manipulation of a device required for fault injection, SCA is based on the passive observation of the target device, e.g., measuring the power consumption or Electro-Magnetic (EM) emanation while the cryptographic algorithm is executed. Using statistical methods, secret information, e.g., a cryptographic key, can be extracted in this way, often without tampering with the device

Based on Technology used:

- *Confidentiality Breach:*
    - Compromise of encryption material
    - Release of Personally identifiable information
- *Counterfeit Hardware:*
    - Compromise of design, manufacture, and/or distribution of information system components including hardware, software and firmware.
    - Counterfeit or tampered-with hardware into the supply chain
- *Data Breach:* Compromise of organizational information systems to facilitate exfiltration of data/information
- *Denial of Service (DoS)*
    - Distributed Denial of Service (DDoS)
    - Simple DoS attack
    - Targeted DoS attack
- *Host Exploit:*
    - Poorly configured or unauthorized information systems exposed to the internet
    - Wireless jamming attacks
- *Inadequate Monitoring of Proximity Events:* Events occurring to any critical infrastructure of the supply chain like airways or storage or delivery systems.
- *Ineffective Disposal:* Obtaining information by opportunistically stealing or scavenging information systems/components.
- *Ineffective testing:*
    - Vulnerabilities in software products
    - Reverse Engineering
    - Software integrity attacks
- *Insider Threat:*
    - A Wave of adaptive and changing cyber-attacks based on detailed surveillance
    - Internal employee coordinating a wave of internal and external attacks across multiple information systems and technologies
    - Coordinate attacks using external, internal and supply chain attack vectors
    - Insert of subverted individuals into organizations
    - Insider-based social engineering to obtain information
    - Vulnerabilities exploited by leveraging internal organization information systems.
- *Malicious Code:*
    - Malicious code delivery to internal organization information systems via email
    - Malware injection using provided removable media
    - Non-targeted zero-day attacks
    - Targeted malware injection into organization information systems and information system components.
    - Untargeted malware injection into downloadable software
- *Phishing:*
    - Creating and operating false front organizations to inject malicious components into supply chain
    - Spear phishing attacks
    - Other phishing attacks
- *Supply Chain Integrity:* Supply chain attacks targeting and exploiting critical hardware, software and firmware
- *Unauthorized access:*
    - Attacks targeting and compromising personal devices of critical employees
    - Attacks using unauthorized ports, protocols and services
    - Brute-force login attempts/password-guessing attacks
    - Communication interception attacks
    - Data-scavenging attacks in a cloud environment
    - Man-in-the-middle attacks

- Counterfeit certificates in SSL communications
- Exploitation of split tunneling
- Externally based session hijacking
- Malicious scanning devices
- Unintended data leaks of sensitive information

Question 3 Part 2 (SQLi):

Given snippet of Code:

```
String query = "select * from employee where employee_name = ?";
List employees = new ArrayList<>();
try (pStatement = con.pStatement(query))
{
        pStatement.setString(1, "rajesh");
        try (ResultSet rSet = pStatement.executeQuery())
        {
                while (rSet.next())
                {
                        employees.add(rSet.getString(1));
                }
        }
}
Assert.assertEquals(1, employees.size());
Assert.assertTrue(employees.contains("rajesh"));

query = "insert into employee(employee_name, ramesh, madhav, vishnu) values(?, ?, ?, ?)"; int
insertedRecordCount;
try (pStatement = con.pStatement(query))
{
        pStatement.setString(1, "ravi");
        pStatement.setInt(2, 239);
        pStatement.setInt(3, 125);
        pStatement.setInt(4, 11);
        insertedRecordCount = pStatement.executeUpdate();
}
Assert.assertEquals(1, insertedRecordCount);

query = "update employee set blue = ? where employee_name = ?";
int updatedRecordCount;
try (pStatement = con.pStatement(query))
{
        pStatement.setInt(1, 10);
        pStatement.setString(2, "orange");
        updatedRecordCount = pStatement.executeUpdate();
}
Assert.assertEquals(1, updatedRecordCount);

query = "delete from employee where employee_name = ?";
int deletedRecordCount;
try (pStatement = con.pStatement(query))
{
        pStatement.setString(1, "ravi");
        deletedRecordCount = pStatement.executeUpdate();
}
Assert.assertEquals(1, deletedRecordCount);
}
```

Assumptions needed to assess the Code:

- ‘con’ is variable containing SQL connection pointer
- ‘pStatement’ is an instance of class PreparedStatements which are used as parameterized queries to define all the SQL code initially and pass in each parameter to the query later.
- ResultSet is instance of Rset class that consists of methods to manipulate set data in webdis/redis stack.
- There are **no** entries in the *employees* database before this snippet executed
- There are columns named *employee_name, blue, ramesh, madhav, Vishnu* (May be they are managers/leads available?) in employees database

Working of Code:

- Operation1: The code is formulating a *preparedStatement* initially to query the *employee* database for values of *employee*_name field.
- Then it is queried for the ‘*rajesh’*. The results are added to an Array named *employees*.
- Then array is checked for its integrity by asserting the array size with 1 and also checking for the string, *‘rajesh’*
- In similar manner (preparedStatement, execute query in try block, assert the values for integrity) the following operations are executed:
  - Operation2: Insert another entry into employee with value of ***employee_name*** **column as *“ravi”, ramesh* column as 239*, madhav* column as 125*, vishnu* column as 11**
  - Operation3: Set value of ***blue*** column to **10** for the employee with ***employee_name*** as ‘**orange’**
  - Operation4: Delete the record of ***employees*** where ***employee_name*** is “***ravi***”

Analysis:

- The code is syntactically correct and would compile successfully
- The SQLi Queries are syntactically correct
- Operation 1 would be successful, and Assertions would be true.
- Operation 2 would be successful, and Assertions would be true.
- Operation 3 would fail, and Assertion would fail as there is no record with ***employee_name*** as ***‘orange'***
- Operation 4 would be successful, and Assertions would be true as the employee created in Operation 2 would be deleted here.
- The code is written in a well-rounded secure manner with
  - prepared statements force the developer to first define all the SQL code, and then pass in each parameter to the query later,
  - all queries executed in try blocks which on failure would not cause exceptions,
  - results checked in assertion statements instead of simple if blocks that would avoid run time errors and is also efficient in detection and correction of bugs.

Question 4:

Prepared Statements:
The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized query. The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once. The prepareStatement() method of Connection interface is used to return the object of PreparedStatement.
public PreparedStatement prepareStatement(String query)throws SQLException{}

The use of prepared statements with variable binding (aka parameterized queries) is how all developers should first be taught how to write database queries. They are simple to write, and easier to understand than dynamic queries. Parameterized queries force the developer to first define all the SQL code, and then pass in each parameter to the query later. This coding style allows the database to distinguish between code and data, regardless of what user input is supplied. Prepared statements ensure that an attacker is not able to change the intent of a query, even if SQL commands are inserted by an attacker

Given snippet:

```
PreparedString div = new PreparedString( "<a href=\"http:\\\\example.com?id=?\"
onmouseover=\"alert('?')\">test</a>", new HTMLEntityCodec() );

div.setURL( 1, request.getParameter( "url" ), new PercentCodec() );
div.set( 2, request.getParameter( "message" ), new JavaScriptCodec() );
out.println( div.toString() );

PreparedString query = new PreparedString(
"SELECT * FROM users WHERE name='?' AND password='?'", new OracleCodec()
);

query.set( 1, request.getParameter( "name" ) );
query.set( 2, request.getParameter( "pass" ) );
stmt.execute( query.toString() );
```

In the above snippet, a parameterized string that uses escaping to make untrusted data safe before combining it with a command or query intended for use in an interpreter is used. Provided the *PreparedString* class is has been included correctly at the start of the string and the respective constructors and methods are defined accurately in the respective headers, the code would compile successfully.

The Codec interface defines a set of methods for encoding and decoding application level encoding schemes, such as HTML entity encoding and percent encoding (aka URL encoding). Codecs are used in output encoding and canonicalization. The design of these codecs allows for character-by-character decoding, which is necessary to detect double-encoding and the use of multiple encoding schemes, both of which are techniques used by attackers to bypass validation and bury encoded attacks in data. **This ensures safety on URL regard as well.**

OracleCodec() is an Implementation of the Codec interface for Oracle strings. This function will only protect you from SQLi in the case of user data bring placed within an Oracle quoted string such as: select * from table where user_name=' USERDATA ';
As a result, there will be no issues in regard to SQL query, as **the Codec would check for possible security concerns when processing the SQL queries for the given input.**

Output of the code: The code would start new instance of HTMLEntityCodec class, sets URL parameter with percent codec to avoid any xss attacks through white space characters and starts a SQL session through Oracle Codec to ensure safe transfer of data.

Question 5:

a) **Worm:** A computer worm is a standalone malware computer program that replicates itself in order to spread to other computers. It often uses a computer network to spread itself, relying on security failures on the target computer to access it. It will use this machine as a host to scan and infect other computers. When these new worm-invaded computers are controlled, the worm will continue to scan and infect other computers using these computers as hosts, and this behaviour will continue. Computer worms use recursive methods to copy themselves without host programs and distribute themselves based on the law of exponential growth, thus controlling and infecting more and more computers in a short time. Worms almost always cause at least some harm to the network, even if only by consuming bandwidth. Worms are also self-replicating in nature, but they don't hook themselves to the program on host computer. They can easily travel from one computer to another if network is available and on the target machine.

b) **Virus:** A computer virus is a type of computer program that, when executed, replicates itself by modifying other computer programs and inserting its own code. Computer viruses generally require a host program. The virus writes its own code into the host program. When the program runs, the written virus program is executed first, causing infection and damage.

They can replicate themselves by hooking them to the program on the host computer like songs, videos etc and then they travel all over the Internet. Examples include File Virus, Macro Virus, Boot Sector Virus, Stealth Virus etc.

c) **Bots:** A bot is a software application that runs automated tasks (scripts) over the Internet. Typically, bots perform tasks that are simple and repetitive much faster than a person could. The most extensive use of bots is for web crawling, in which an automated script fetches, analyzes and files information from web servers. They are automated processes that are designed to interact over the internet without the need of human interaction. They can be good or bad. Malicious bot can infect one host and after infecting creates connection to the central server which will provide commands to all infected hosts attached to that network called Botnet.

d) **DDoS:** A distributed denial-of-service (DDoS) attack occurs when multiple systems flood the bandwidth or resources of a targeted system, usually one or more web servers. A DDoS attack uses more than one unique IP address or machines, often from thousands of hosts infected with malware. A distributed denial of service attack typically involves more than around 3–5 nodes on different networks; fewer nodes may qualify as a DoS attack but is not a DDoS attack.

Multiple machines can generate more attack traffic than one machine, multiple attack machines are harder to turn off than one attack machine, and that the behaviour of each attack machine can be stealthier, making it harder to track and shut down. Since the incoming traffic flooding the victim originates from different sources, it may be impossible to stop the attack simply by using ingress filtering. It also makes it difficult to distinguish legitimate user traffic from attack traffic when spread across multiple points of origin.

e) **XSS:** Cross-site scripting (XSS) is a type of security vulnerability that can be found in some web applications. XSS attacks enable attackers to inject client-side scripts into web pages viewed by other users. A cross-site scripting vulnerability may be used by attackers to bypass access controls such as the same-origin policy. XSS effects vary in range from petty nuisance to significant security risk, depending on the sensitivity of the data handled by the vulnerable site and the nature of any security mitigation implemented by the site's owner network. There are two primary flavours of XSS flaws: non-persistent and persistent. These can further be divided into two groups: traditional (caused by server-side code flaws) and DOM-based (in client-side code).

Question 6:

Although it is possible to determine if a program or a code snippet is 'safe' to run, it is increasingly difficult to the accuracy of the results. The best one can do is to execute the part of script using an agent that is regarded safe in precursor.

The given program is:

```
Program CV :=
{....
        main-program :=
        {
                if D(CV) then goto next:
                else infect-executable;
        }
        next:
}
```

- In the above snippet, we can see that the D is a program that checks other programs such as CV and returns TRUE if the program given as an argument (CV in our case) is a computer virus and it returns FALSE if it is not a computer virus.
- In the case that CV is a program containing a virus, D simply progresses to the 'next' phase. As a result, the virus in CV program will not infect the executable programs.
- On the other hand, if D results FALSE for CV program, then it means that CV can infect the executable programs. Hence the given condition in the program is wrong.