**Software development security**

Computers understand binary code. They speak a language of 1s and 0s, and that’s

it! The instructions that a computer follows consist of a long series of binary digits in a

language known as *machine language.*

Programmers don’t want to write their code in either machine language or assembly language. They prefer to use high-level languages, such as C++, Ruby, Java, and Visual Basic

Once programmers are ready to execute their programs, two options are available to them: **compilation and interpretation.**

**C, JAVA and FORTRAN are complied language**. Uses complier to convert from high level language into an executable file. This executable is then distributed to end users, who may use it as they see fi t. Generally speaking, it’s not possible to view or modify the software instructions in an executable fi le

**JavaScript, VBSCRIPT are interpreted languages**. When these languages are used, the programmer distributes the source code, which contains instructions in the higher-level language. **End users then use an interpreter** to execute that source code on their systems. They’re able to view the original instructions written by the programmer

**First-generation languages (1GL) include all machine languages.**

**■ Second-generation languages (2GL) include all assembly languages.**

**■ Third-generation languages (3GL) include all compiled languages.**

**■ Fourth-generation languages (4GL) attempt to approximate natural languages and**

**include SQL, which is used by databases.**

**■ Fifth-generation languages (5GL) allow programmers to create code using visual**

**interfaces.**

From a security point of view, **object-oriented programming provides a black-box approach to abstraction.**

**Cohesion** describes the strength of the relationship between the purposes of the methods within the same class.

**Coupling** is the level of interaction between objects. Lower coupling means less interaction. Lower coupling provides better software design because objects are more independent. Lower coupling is easier to troubleshoot and update. Objects that have low cohesion require lots of assistance from other objects to perform tasks and have high coupling.

**Assurance**

To ensure that the security control mechanisms built into a new application properly implement the security policy throughout the life cycle of the system, administrators use *assurance procedures* . Assurance procedures are simply formalized processes by which trust is built into the life cycle of a system. The **Trusted Computer System Evaluation Criteria** (TCSEC) Orange Book refers to this process as ***life cycle assurance****.*

***Input validation***verifies that the values provided by a user match the programmer’s expectation

before allowing further processing. For example, input validation would check whether a month value is an integer between 1 and 12. If the value falls outside that range, the program will not try to process the number as a date and will inform the user of the input expectations. **This type of input validation, where the code checks to ensure that a number falls within an acceptable range, is known as a *limit check* .**

In some cases, the input validation routine can transform the input **to remove risky character sequences and replace them with safe values. This process is known as escaping input.**

**Input validation should always occur on the server side of the transaction.**

The ***fail-secure failure state***puts the system into **a high level of security** (and possibly

even disables it entirely) until an administrator can diagnose the problem and restore

the system to normal operation.

**The *fail-open state***allows users **to bypass failed security controls**, erring on the side of

permissiveness.

**Systems development life cycle**

These core activities are essential to the development of sound, secure systems:

■ **Conceptual definition**

The conceptual definition phase of systems development involves creating the basic concept statement for a system. It’s a simple statement agreed on by all interested stakeholders (the developers, customers, and management) that states the purpose of the project as well as the general system requirements. **The conceptual definition is a very high-level statement of purpose and should not be longer than one or two paragraphs.** It’s very helpful to **refer to the concept statement** at all phases of the systems development process. Simply reading the concept statement periodically can assist in refocusing a team of developers.

■ **Functional requirements determination**

Once all stakeholders have agreed on the concept statement, it’s time for the development team to sit down and begin the functional requirements process. In this phase, specific system functionalities are listed. In the final stages of testing and evaluation, the project managers should use this document as a checklist to ensure that all functional requirements are met.

**■ Control specifications development**

**Security-conscious organizations also ensure that adequate security controls are designed** into every system from the earliest stages of development. It’s often useful **to have a control specifications development phase in your life cycle model**. This phase takes place soon after the development of functional requirements and often continues as the design and design review phases progress.

■ **Design review**

Once the functional and control specifications are complete, let the system designers do their thing! In this often-lengthy process, the designers determine exactly how the various parts of the system will interoperate and how the modular system structure will be laid out. Also, during this phase the design management team commonly sets specific tasks for various teams and lays out initial timelines for the completion of coding milestones.

■ **Code review walk-through**

Once the stakeholders have given the software design their blessing, it’s time for the software developers to start writing code. Project managers should schedule several code review walk-through meetings at various milestones throughout the coding process. These technical meetings usually involve only development personnel. **looking for problems in logical flow or other design/security flaws.**

■ **System test review /User Acceptance Testing**

Now it’s time **to begin the user acceptance testing** phase. Initially, most organizations perform the initial system tests using development personnel to seek out any obvious errors. As the testing progresses, developers and actual users validate the system against predefined scenarios that model common and unusual user activities. Once this phase is complete, the code may move to deployment. As with any critical development process**, it’s important that you maintain a copy of the written test plan and test results for future review.**

■ **Maintenance and change management**

Once a system is operational, a variety of maintenance tasks are necessary to ensure continued operation in the face of changing operational, data processing, storage, and environmental requirements. It’s essential that you have a skilled support team in place to handle any routine or unexpected maintenance. It’s also important that any changes to the code be handled through a formalized change management process,

----------------------------------------------------

**In 1991, the Software Engineering Institute introduced the Capability Maturity Model**, which described the process organizations undertake as they move toward incorporating solid engineering principles into their software development processes.

**Waterfall model**

**7 stages**. As each stage is completed, the project moves into the next phase. the modern waterfall model **does allow development to return to the previous phase to correct defects** discovered during the subsequent phase. This is often known as **the *feedback loop characteristic* of the waterfall model.**

**1.System requirements, 2. Software requirements,3. preliminary design,4. detailed design,5.code and debug,6.testing,7.operations and maintenance**

The waterfall model was one of the **first comprehensive attempts** to model the software development process. major criticisms of this model are that it allows the developers to step back only one phase in the process. **It does not make provisions for the discovery of errors at a later** phase in the development cycle. The waterfall model was improved by adding **validation and verification**

steps to each phase**. The improved model was labeled the *modified* waterfall model.**

**Spiral Model**

It allows for **multiple iterations of a waterfall-style process**. the spiral model **encapsulates a number of iterations of another model (the waterfall model), it is known as a *metamodel,* or a “model of models**

**1.Determine objectives, alternatives, and constraints2. Evaluate alternatives Identify and resolve risks,3. Develop and verify next-level product.4. Plan next phase**

**it allows developers to return to the planning stages as changing technical demands and customer requirements necessitate the evolution of a system.**

**Agile Software Development**

The core philosophy of the Agile approach:

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

**Individuals and interactions** over processes and tools. **Working software** over comprehensive documentation. **Customer collaboration** over contract negotiation. **Responding to change** over following a plan.

The *Agile Manifesto* also defines **12 principles that underlie the philosophy**

■ Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

■ Welcome changing requirements, even late in development. Agile processes harness change for the customer’s competitive advantage.

■ Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

■ Business people and developers must work together daily throughout the project.

■ Build projects around motivated individuals. Give them the environment and support they need and trust them to get the job done.

■ **The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.**

■ Working software is the primary measure of progress.

■ **Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.**

■ **Continuous attention to technical excellence and good design enhances agility.**

■ **Simplicity**—the art of maximizing the amount of work not done—is essential.

■ The best architectures, requirements, and designs emerge from self-organizing teams.

■ **At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.**

**The Agile development approach is quickly gaining momentum in the software community and has many variants, including Scrum, Agile Unified Process (AUP), the Dynamic Systems Development Model (DSDM), and Extreme Programming (XP).**

**Software Capability Maturity Model**

The idea behind the SW-CMM is that **the quality of software depends on the quality** of its development process

**Level 1: Initial** In this phase, you’ll often find hardworking people charging ahead in a disorganized fashion. There is usually little or no defined software development process.

**Level 2: Repeatable** In this phase, basic life cycle management processes are introduced. **Reuse of code** in an organized fashion begins to enter the picture, and repeatable results are expected from similar projects. SEI defines the **key process areas for this level as Requirements Management, Software Project Planning, Software Project Tracking and Oversight, Software Subcontract Management, Software Quality Assurance, and Software Configuration Management**.

**Level 3: Defined** In this phase, software developers **operate according to a set of formal, documented software development processes**. All development projects take place within the constraints of the new standardized management model. SEI defines the **key process areas for this level as Organization Process Focus, Organization Process Definition, Training Program, Integrated Software Management, Software Product Engineering, Intergroup Coordination, and Peer Reviews.**

**Level 4: Managed** In this phase, management of the software process proceeds to the next level**. Quantitative measures are utilized to gain a detailed understanding of the development process. SEI defines the key process areas for this level as Quantitative Process Management and Software Quality Management.**

**Level 5: Optimizing** In the optimized organization, a process **of continuous improvement** occurs. Sophisticated software development processes are in place that ensure that feedback from one phase reaches to the previous phase to improve future results. SEI defines the **key process areas for this level as Defect Prevention, Technology Change Management, and Process Change Management**

**IDEAL Model**

The Software Engineering Institute also developed the IDEAL model for software development, which implements many of the SW-CMM attributes. The IDEAL model has five phases:

**I: Initiating** In the initiating phase of the IDEAL model, the business reasons behind the change are outlined, support is built for the initiative, and the appropriate infrastructure is put in place.

**2: Diagnosing** During the diagnosing phase, engineers analyze the current state of the organization and make general recommendations for change.

**3: Establishing** In the establishing phase, the organization takes the general recommendations from the diagnosing phase and develops a specific plan of action that helps achieve those changes.

**4: Acting** In the acting phase, it’s time to stop “talking the talk” and “walk the walk.” The organization develops solutions and then tests, refines, and implements them.

**5: Learning** As with any quality improvement process, the organization must continuously analyze its efforts to determine whether it has achieved the desired goals and, when necessary, propose new actions to put the organization back on course.

**Easy to remember ideal and swcmm I…I, Dr. Ed, am lo(w).”**

**Gantt Charts**

A Gantt chart is a type of bar chart that shows the interrelationships over time between projects and schedules. It provides a graphical illustration of a schedule that helps to plan, coordinate, and track specific tasks in a project

**PERT**

Program Evaluation Review Technique (PERT) is a **project-scheduling tool used to judge the size of a software product in development and calculate the standard deviation (SD) for risk assessment**. PERT relates the estimated lowest possible size, the most likely size, and the highest possible size of each component. PERT is used to direct improvements to project management and software coding to produce more efficient software**. As the capabilities of programming and management improve, the actual produced size of software should be smaller**

**Change management (also known as control management**) plays an important role when monitoring systems in the controlled environment of a datacenter. 1. **Request control (can conduct cost benefit analysis). 2. Change control. 3.Release control (**Release control should also include acceptance testing to ensure that any alterations to end-user work tasks are understood and functional**)**

**configuration management**

**Configuration Identification** During the configuration identification process, administrators document the configuration of covered software products throughout the organization.

**Configuration Control The configuration control process ensures that changes to software versions are made in accordance with the change control and configuration** **management policies**. **Updates can be made only from authorized distributions in accordance with those policies.**

**Configuration Status Accounting** Formalized procedures are used to keep track of all authorized changes that take place.

**Configuration Audit** A periodic configuration audit should be conducted to ensure that the actual production environment is consistent with the accounting records and that no unauthorized configuration changes have taken place

**The DevOps Approach**

**Software development, operations and quality assurance combined is called devops approach.**

**Application programming interface(API)**

APIs must also be tested thoroughly for security flaws, just like any web application.

**Software Testing**

One of the tests you should perform is a ***reasonableness check.*** The reasonableness check ensures that values returned by software match specified criteria that are within reasonable bounds. For example, a routine that calculated optimal weight for a human being and returned a value of 612 pounds would certainly fail a reasonableness check!

When testing software, you should apply the same rules of separation of duties that you do for other aspects of your organization. In other words, you should assign the testing of your software to someone other than the programmer(s) who developed the code to avoid a conflict of interest and assure a more secure and functional finished product.

**White-box Testing** White-box testing examines the internal logical structures of a program and steps through **the code line by line, analyzing the program for potential errors.**

**Black-box Testing** Black-box testing examines the program from a **user perspective by providing** a wide variety of input scenarios and inspecting the output. Black-box **testers do not have access to the internal** code**. Final acceptance testing that occurs prior to system delivery is a common example of black-box testing**.

**Gray-box Testing** Gray-box **testing combines the two approaches and is popular for software validation**. In this approach, testers examine the software from a user perspective, analyzing inputs and outputs. They also **have access to the source code** and use it to help design their tests. They do not, however, analyze the inner workings of the program during their testing.

**There are two categories of testing used specifically to evaluate application security:**

**Static Testing** Static testing evaluates the **security of software without running** it by analyzing either the source code or the compiled application. Static analysis usually involves the use **of automated tools** designed to detect common software flaws, such as buffer overflows. In mature development environments, application developers are given access to static analysis tools and use them throughout the design/build/test process.

**Dynamic Testing** Dynamic testing evaluates the security of software in a **runtime environment and is often the only option for organizations deploying applications written by someone else. I**n those cases, testers often do not have access to the underlying source code. One common example of dynamic software testing is the use of web application scanning tools to detect the presence of cross-site scripting, SQL injection, or other flaws in web applications. Dynamic tests on a production environment should always be carefully coordinated to avoid an unintended interruption of service.

Proper software test implementation is a key element in the project development process

**code repositories** such as **GitHub, Bitbucket, and SourceForge** also provide version control, bug tracking, web hosting, release management, and communications functions that support software development

**SLA**

It covers 1. System uptime (as a percentage of overall operating time) Maximum consecutive downtime (in seconds/minutes/and so on) ■ Peak load■ Average load, Responsibility for diagnostics ■ Failover time (if redundancy is in place)

**Software acquisition**

In the case of SaaS environments, most security responsibility rests with the vendor, but the organization’s security staff isn’t off the hook. Although they might not be responsible for as much configuration, they now take on responsibility for monitoring the vendor’s security. This may include audits, assessments, vulnerability scans, and other measures designed to verify that the vendor maintains proper controls.

**Database Management System Architecture**

DBMS architectures: hierarchical and distributed

**A hierarchical data model** combines records and fields that are related in a logical tree structure. This results in a **one-to-many data** model. One manager can have multiple employees but one employee can have one manager

**The distributed data model** has data stored in more than one database, but those databases are logically connected. The user perceives the database as a single entity, even though it consists of numerous parts interconnected over a network. Each field can have numerous children as well as numerous parents. Thus, the data mapping relationship for distributed databases is **many-to-many.**

**A relational database** consists of flat two-dimensional tables made up of rows and columns. In fact, each table looks like a spreadsheet fi le. The row and column structure provide for one-to-one data mapping relationships

True object-oriented databases (OODBs) benefit t from ease of code reuse, ease of troubleshooting analysis, and reduced overall maintenance. OODBs are also better suited than other types of databases for supporting complex applications involving multimedia, CAD, video, graphics, and expert systems

Each customer would have its own **record, or *tuple***, represented by **a row** in the table. **The number of rows in the relation is referred to as *cardinality*, and the number of columns is the *degree***

*keys* are a subset of the fields of a table and are used to uniquely identify records

**Candidate Keys** A *candidate key* is a subset of attributes that can be used to uniquely identify any record in a table. No two records in the same table will ever contain the same values for all attributes composing a candidate key. Each table may have one or more candidate keys, which are chosen from column headings.

**Primary Keys** A *primary key* is selected from the set of candidate keys for a table to be used to uniquely identify the records in a table. Each table has only one primary key, selected by the database designer from the set of candidate keys. **The RDBMS enforces the uniqueness of primary keys by disallowing the insertion of multiple records with the same primary key. I**n the Customers table shown in Figure 20.8 , the Customer ID would likely be the primary key.

**Foreign Keys** A *foreign key* is used to enforce relationships between two tables, also known as *referential integrity* . Referential integrity ensures that if one table contains a foreign key, it corresponds to a still-existing primary key in the other table in the relationship. It makes certain that no record/tuple/row contains a reference to a primary key of a nonexistent record/tuple/row.

Database developers strive to create well-organized and efficient databases. To assist with this effort, they’ve defined several levels of database organization known as *normal forms* . The process of bringing a database table into compliance with normal forms is known as ***normalization***

SQL itself is divided into two distinct components: the Data **Definition Language (DDL), which allows for the creation and modification of the database’s structure (known as the *schema* ),** and the Data Manipulation Language (DML), which allows users to interact with the data contained within that schema.

All database transactions have four required characteristics: atomicity, consistency, isolation,

and durability. Together, these attributes are known as the *ACID model* , which is a

critical concept in the development of database management systems. Let’s take a brief look

at each of these requirements:

**Atomicity** Database transactions must be atomic—that is, they must be an “all-or nothing”

affair. If any part of the transaction fails, the entire transaction must be rolled back as if it never occurred.

**Consistency** All transactions must begin operating in an environment that is consistent with all of the database’s rules (for example, all records have a unique primary key). When the transaction is complete, the database must again be consistent with the rules, regardless of whether those rules were violated during the processing of the transaction itself. No other transaction should ever be able to use any inconsistent data that might be generated during the execution of another transaction.

**Isolation** The isolation principle requires that transactions operate separately from each other. If a database receives two SQL transactions that modify the same data, one transaction must be completed in its entirety before the other transaction is allowed to modify the same data. This prevents one transaction from working with invalid data generated as an intermediate step by another transaction.

**Durability** Database transactions must be durable. That is, once they are committed to the database, they must be preserved. Databases ensure durability through the use **of backup mechanisms, such as transaction logs**

**When multilevel security is required, it’s essential that administrators and developers strive to keep data with different security requirements separate. Mixing data with different classification** levels and/or need-to-know requirements is known as ***database contamination***and is a significant security challenge

Another way to implement multilevel security in a database is through the use of database views. Views are simply SQL statements that present data to the user as if the views were tables themselves. Views may be used to collate data from multiple tables, aggregate individual records, or restrict a user’s access to a limited subset of database attributes and/or records.

**Views are stored in the database as SQL commands rather than as tables of data**. This dramatically reduces the space requirements of the database and allows views to violate the rules of normalization that apply to tables. However, retrieving data from a complex view can take significantly longer than retrieving it from a table because the DBMS may need to perform calculations to determine the value of certain attributes for

each record. Because views are so flexible, many database administrators use them as a security

tool—allowing users to interact only with limited views rather than with the raw tables of

data underlying them.

**Concurrency**

***Concurrency,* or edit control, is a preventive security mechanism that endeavors to make certain that the information stored in the database is always correct or at least has its integrity and availability protected**. This feature can be employed on a single-level or multilevel database. **Concurrency uses a “lock” feature** to allow one user to make changes but deny other users access to views or make changes to data elements at the same time. Then, after the changes have been made, an “unlock” feature restores the ability of other users to access the data they need**. In some instances, administrators will use concurrency with auditing mechanisms to track document and/or field changes. When this recorded data is reviewed, concurrency becomes a detective control.**

**Semantic integrity** ensures that user actions don’t violate any structural rules. It also checks that all stored data types are within valid domain ranges, ensures that only logical values exist, and confirms that the system complies with all uniqueness constraints.

**Administrators may employ time and date stamps to maintain data integrity and availability. Time and date stamps often appear in distributed database systems. When a time stamp is placed on all change transactions and those changes are distributed or replicated to the other database members, all changes are applied to all members, but they are implemented in correct chronological order**.

Another common security feature of a DBMS is that objects can be controlled granularly within the database; this can also improve security control. Content-dependent access control is an example of granular object control. **Content-dependent access control is based on the contents or payload of the object being accessed. Because decisions must be made on an object-by-object basis, content-dependent control increases processing overhead**. **Another form of granular control is *cell suppression****.* **Cell suppression is the concept of hiding individual database fields or cells or imposing more security restrictions on them**.

Context-dependent access control is often discussed alongside content-dependent access control because of the similarity of the terms. **Context-dependent access control evaluates the big picture to make access control decisions. The key factor in context-dependent access control is how each object or packet or field relates to the overall activity or communication.** Any single element may look innocuous by itself, but in a larger context that element may be revealed to be benign or malign. Administrators might employ database partitioning to subvert aggregation and inference vulnerabilities.

**Database partitioning is the process of splitting a single database into multiple parts, each with a unique and distinct security level or type of content**

***Polyinstantiation* occurs when two or more rows in the same relational database table appear to have identical primary key elements but contain different data for use at differing classification levels. It is often used as a defense against some types of inference attacks**

*Primary (or “real”) memory* consists of the main memory resources directly available to a system’s CPU. Primary memory normally consists of **volatile** random access memory(RAM) and is usually the most high-performance storage resource available to a system.

*Secondary storage* consists of more inexpensive, nonvolatile storage resources available to a system for long-term use. Typical secondary storage resources include magnetic and optical media, such as tapes, disks, hard drives, flash drives, and CD/DVD storage.

*Virtual memory* allows a system to simulate additional primary memory resources through the use of secondary storage. For example, a system low on expensive RAM might make a portion of the hard disk available for direct CPU addressing.

*Virtual storage* allows a system to simulate secondary storage resources through the use of primary storage. The most common example of virtual storage is the RAM disk that presents itself to the operating system as a secondary storage device but is actually implemented in volatile RAM. This provides an extremely fast filesystem for use in various applications but provides no recovery capability.

*Random access storage* allows the operating system to request contents from any point within the media. RAM and hard drives are examples of random access storage resources.

*Sequential access storage* requires scanning through the entire media from the beginning to reach a specific address. **A magnetic tape is a common example of a sequential access storage resource.**

*Volatile storage* loses its contents when power is removed from the resource**. RAM is the most common type of volatile storage resource.**

*Nonvolatile storage* does not depend upon the presence of power to maintain its contents. Magnetic/optical media and nonvolatile RAM (NVRAM) are typical examples of nonvolatile storage resources.

First, the threat of illegitimate access to storage resources exists no matter what type of storage is in use

Second, Covert channel attacks pose the second primary threat against data storage resources. **Covert storage channels allow the transmission of sensitive data between classification levels through the direct or indirect manipulation of shared storage media.**

**Two types of knowledge-based artificial intelligence systems: expert systems and neural networks**

**Every expert system has two main components .The knowledge base and the inference engine.** Expert systems are not infallible—they’re only as good as the data in the knowledge base and the decision-making algorithms implemented in the inference engine. However, they have one major advantage in stressful situations—their decisions do not involve judgment clouded by emotion. Expert systems can play an important role in analyzing emergency events, stock trading, and other scenarios in which emotional investment sometimes gets in the way of a logical decision. For this reason, many lending institutions now use expert systems to make credit decisions instead of relying on loan officers who might say to themselves,

**Fuzzy logic**

As previously mentioned, inference engines commonly use a technique known as *fuzzy logic* . This technique is designed to more closely approximate human thought patterns than the rigid mathematics of set theory or algebraic approaches that use “black-andwhite” categorizations of data. Fuzzy logic replaces them with blurred boundaries, allowing the algorithm to think in the “shades of gray” that dominate human thought. Fuzzy logic as used by an expert system has four steps or phases: **fuzzification, inference, composition,and defuzzification.**

**Neural Networks**

In neural networks, chains of computational units are used in an attempt to imitate the biological reasoning process of the human mind. In an expert system, a series of rules is stored in a knowledge base, whereas in a neural network, a long chain of computational decisions that feed into each other and eventually sum to produce the desired output is

set up.

These benefits are evident in the implementations of neural networks for voice recognition, face recognition, weather prediction, and the exploration of models of thinking and consciousness

**Decision Support Systems**

A *decision support system (DSS)* is a knowledge-based application that analyzes business data and presents it in such a way as to make business decisions easier for users**. It is considered more of an informational application than an operational application**. Often a DSS is employed by knowledge workers (such as help desk or customer support personnel) and by sales services (such as phone operators). This type of application may present

information in a graphical manner to link concepts and content and guide the script of the operator. Often a DSS is backed by an expert system controlling a database.

**Security Applications**

Both expert systems and neural networks have great applications in the field of computer security. One of the major advantages offered by these systems is their capability to rapidly make consistent decisions. One of the major problems in computer security is the inability of system administrators to consistently and thoroughly analyze massive amounts of log and audit trail data to look for anomalies. It seems like a match made in heaven! One successful application of this technology to the computer security arena is the Next- Generation Intrusion Detection Expert System (NIDES) developed by Phillip Porras and his team at the Information and Computing Sciences System Design Laboratory of SRI International. This system provides an inference engine and knowledge base that draws information from a variety of audit logs across a network and provides notification to security administrators when the activity of an individual user varies from the user’s standard usage profile.

* Unit – You test each individual piece of code. Think each class or method.
* Integration – You test the integrations of many units together. You make sure your code works when put together, including dependencies, databases and libraries.
* Regression – After integrating (and maybe fixing) you should run your unit tests again. This is regression testing to ensure that further changes have not broken any units that were already tested. You can run your unit tests again and again for regression testing.
* Acceptance – You should test that the program works the way a user/customer expects the application to work. Acceptance tests ensure that the functionality meets business requirements.
* Spyware- Spyware is any technology that aids in gathering information about a person or organization without their knowledge. On the Internet (where it is sometimes called a Spybot or tracking software), Spyware is programming that is put in someone's computer to secretly gather information about the user and relay it to advertisers or other interested parties. Spyware can get in a computer as a software virus or as the result of installing a new program.
* Virus- a virus is a program or programming code that replicates by being copied or initiating its copying to another program, computer boot sector or document. Viruses can be transmitted as attachments to an e-mail note or in a downloaded file, or be present on a diskette or CD
* Worm- a worm is a self-replicating virus that does not alter files but duplicates itself. It is common for worms to be noticed only when their uncontrolled replication consumes system resources, slowing or halting other tasks.
* Logic bomb- a logic bomb is programming code, inserted surreptitiously or intentionally, that is designed to execute (or "explode") under circumstances such as the lapse of a certain amount of time or the failure of a program user to respond to a program command. It is in effect a delayed-action computer virus or Trojan horse. A logic bomb, when "exploded," may be designed to display or print a spurious message, delete or corrupt data, or have other undesirable effects.
* Trapdoor- is a method of gaining access to some part of a system other than by the normal procedure (e.g. gaining access without having to supply a password). Hackers who successfully penetrate a system may insert trapdoors to allow them entry at a later date, even if the vulnerability that they originally exploited is closed. There have also been instances of system developers leaving debug trapdoors in software, which are then discovered and exploited by hackers.
* Trojan (Trojan Horse)- a Trojan horse is a program in which malicious or harmful code is contained inside apparently harmless programming or data in such a way that it can get control and do its chosen form of damage, such as ruining the certain area on your hard disk. A Trojan horse may be widely redistributed as part of a computer virus.
* RATs (Remote Admin Trojans) - are a special form of Trojan Horse that allows remote control over a machine. These programs are used to steal passwords and other sensitive information. Although they are "invisible", symptoms such as a slow moving system, CD ports opening and closing and unexplained restarting of your computer may manifest.
* Malware - Malware (for "malicious software") is any program or file that is harmful to a computer user. Thus, malware includes computer viruses, worms, Trojan horses, and also Spyware, programming that gathers information about a computer user without permission.
* Mobile Malicious Code - web documents often have server-supplied code associated with them which executes inside the web browser. This active content allows information servers to customize the presentation of their information, but also provides a mechanism to attack systems running a client browser. Mobile malicious code may arrive at a site through active content such as JavaScript, Java Applets and ActiveX controls or through Plug-ins.
* Malicious Font - webpage text that exploits the default method used to de-compress Embedded Open Type Fonts in Windows based programs including Internet Explorer and Outlook. These malicious fonts are designed to trigger a buffer overflow which will disable the security on Windows-based PCs. This allows an intruder to take complete control of the affected computer and remotely execute destructive activities including installing unauthorized programs and manipulating data.
* Rootkits - Rootkits are a set of software tools used by an intruder to gain and maintain access to a computer system without the user's knowledge. These tools conceal covert running processes, files and system data making them difficult to detect. There are rootkits to penetrate a wide variety of operating systems including Linux, Solaris and versions of Microsoft Windows. A computer with rootkits on it is called a rooted computer.

There are three types of rootkits. Below is a description of the characteristics of each:

* + Kernel Rootkits - hide a backdoor on a computer system by using modified code to add or replace a portion of the system's existing kernel code. Usually the new code is added to the kernel via a device driver or loadable module. Kernel rootkits can be especially dangerous because they can be difficult to detect without appropriate software.
  + Library Rootkits - hide information about the intruder by manipulating system calls with patches, hooks, or replacements.
  + Application Rootkits - replace or modify regular application binaries with camouflaged fakes, hooks, patches, or injected code..

**..**