**OPERATING SYSTEM ASSIGNMENT**

**Submitted by:**

**(102103769) RACHIT BHAGAT**

**BE Second Year (<C0E27>)**

Submitted to: Dr Shashank S Singh



# Computer Science and Engineering Department

**Thapar Institute of Engineering and Technology**

# November 2022

1. **Define various aspects of Security of an OS and list various security attack/Program threat.**

Security deals with protecting systems from deliberate attacks either internal or external, from individuals intentionally attempting to steal information, damage information or otherwise creating great damage in some manner. The operating system must be protected from security breaches, such as runaway processes (DOS), memory-access violations, stack overflow violations, the launching of programs with excessive privileges and many other.[1] The term operating system security refers to practices and measures that can ensure the confidentiality, integrity and availability (CIA) of operating systems. The goal of OS security is to protect the OS from various threats, including malicious software such as worms, trojans and other viruses, misconfigurations and remote intrusions. OS security typically involves the implementation of control techniques that can protect your assets from unauthorized modification and deletion or theft. The most common techniques used to protect operating systems include the use of antivirus software and other endpoint protection measures, regular OS patch updates, a firewall for monitoring network traffic and enforcement of secure access through least privileges and user controls.[2]

There are many common program threats to modern systems.

* + Trojan Horse: A Trojan Horse is a program that secretly performs some maliciousness in addition to its visible actions. Some Trojan horses are deliberately written such, and others are result of legitimate programs that have become infected with viruses. One dangerous opening for Trojan horses is long search paths, and in particular paths which include the current directory (“.”) as part of the path. Another classic Trojan Horse is a login emulator, which records users account name and password issues a “password incorrect” message, and then logs off the system. Then user tries again and logs in successfully, and doesn’t realize that their information has been stolen. Two solutions to Trojan Horse are to have the system print usage statics on logouts, and to require the typing of non-trappable key sequences such as Control-Alt-Delete in order to login.[1]
  + Trap Door: A Trap Door is a when programmer/hacker deliberately inserts a security hole that they can use later to access the system. Because of the possibility of trap doors, once a system has been in an untrustworthy state, that system can never been trusted again. Even the backup tapes may contain a copy of some cleverly hidden back door. A clever trap door could be inserted into a compiler so that compiler would contain a security hole. This is especially dangerous, because inspection of the code being compiled would not reveal any problems.[1]
  + Logic Bomb: A Logic Bomb is code that is not designed to cause havoc all the time, but only when a certain set of circumstances occurs, such as when a particular date or time is reached or some or some other noticeable event. A classic example is the Dead-Man Switch, which is designed to check whether a certain person is logging in everyday, and if they don’t login for a long time, then logic bomb goes off and either opens up security holes or cause other problems.[1]
  + Stack and Buffer overflow: In software, a stack buffer overflow or stack buffer overrun occurs when a program writes to a memory address on the program's call stack outside of the intended data structure, which is usually a fixed-length buffer. Stack buffer overflow bugs are caused when a program writes more data to a buffer located on the stack than what is actually allocated for that buffer. This almost always results in corruption of adjacent data on the stack, and in cases where the overflow was triggered by mistake, will often cause the program to crash or operate incorrectly. Stack buffer overflow is a type of the more general programming malfunction known as buffer overflow (or buffer overrun). Overfilling a buffer on the stack is more likely to derail program execution than overfilling a buffer on the heap because the stack contains the return addresses for all active function calls. A stack buffer overflow can be caused deliberately as part of an attack known as stack smashing. If the affected program is running with special privileges, or accepts data from untrusted network hosts then the bug is a potential security vulnerability. This is one of the oldest and more reliable methods for attackers to gain unauthorized access to a computer.[3]
  + Viruses: A virus is a fragment of code embedded in an otherwise legitimate program, designed to replicate itself, and wreaking havoc. Viruses are more likely to infect PCs than UNIX or other multi-user systems, because programs in the latter systems have limited authority to modify other programs or to access critical system structures. Viruses are delivered to systems in a virus dropper, usually some form of Trojan Horse and usually via email or unsafe downloads.[1]

1. **Explain various categories of viruses related to Operating System.**

A virus is a fragment of code embedded in a legitimate program. Viruses are self-replicating and are designed to infect other programs. They can wreak havoc in a system by modifying or destroying files causing system crashes and program malfunctions. On reaching the target machine a virus dropper (usually a trojan horse) inserts the virus into the system.

Various types of viruses:

* File virus: This type of virus infects the system by appending itself to the end of a file. It changes the start of a program so that the control jumps to its code. After the execution of its code, the control returns back to the main program. Its execution is not even noticed. It is also called a Parasitic virus because it leaves no file intact but also leaves the host functional.
* Boot sector virus: It infects the boot sector of the system, executing every time system is booted and before the operating system is loaded. It infects other bootable media like floppy disks. These are also known as memory viruses as they do not infect the file systems.
* Macro virus: Unlike most viruses which are written in a low-level language(like C or assembly language), these are written in a      high-level language like Visual Basic. These viruses are triggered when a program capable of executing a macro is run. For example, the macro viruses can be contained in spreadsheet files.
* Polymorphic virus: A virus signature is a pattern that can identify a virus (a series of bytes that make up virus code). In order to avoid detection by antivirus a polymorphic virus changes each time it is installed. The functionality of the virus remains the same but its signature is changed.
* Encrypted virus: In order to avoid detection by antivirus, this type of virus exists in encrypted form. It carries a decryption algorithm along with it. The virus first decrypts and then executes.
* Stealth virus: It is a very tricky virus as it changes the code that can be used to detect it. Hence, the detection of viruses becomes very difficult. For example, it can change the read system call such that whenever the user asks to read a code modified by a virus, the original form of code is shown rather than infected code.
* Tunneling virus: This virus attempts to bypass detection by antivirus scanner by installing itself in the interrupt handler chain. Interception programs, which remain in the background of an operating system and catch viruses, become disabled during the course of a tunneling virus. Similar viruses install themselves in device drivers.
* Multipartite virus: This type of virus is able to infect multiple parts of a system including the boot sector, memory, and files. This makes it difficult to detect and contain.
* Armored virus: An armored virus is coded to make it difficult for antivirus to unravel and understand. It uses a variety of techniques to do so like fooling antivirus to believe that it lies somewhere else than its real location or using compression to complicate its code.
* Browser Hijacker: As the name suggests this virus is coded to target the user’s browser and can alter the browser settings. It is also called the browser redirect virus because it redirects your browser to other malicious sites that can harm your computer system.
* Resident virus: Resident viruses installation store for your RAM and meddle together along with your device operations. They’re so sneaky that they could even connect themselves for your anti-virus software program files.[4]

**3. Describe the various Security Defences including firewall systems to protect systems and network.**

* + Cryptography as Security tool: Within a given computer the transmittal of messages is safe, reliable and secure, because the OS knows exactly where each one is coming from and where it is going. A rogue computer (or e-mail sender) may spoof their identity, and outgoing packets are delivered to a lot of other computers besides their final destination, which brings up two big questions of security:

Trust - How can the system be sure that the messages received are really from the source that they say they are, and can that source be trusted?

Confidentiality - How can one ensure that the messages one is sending are received only by the intended recipient?

Cryptography can help with both of these problems, through a system of secrets and keys. In the former case, the key is held by the sender, so that the recipient knows that only the authentic author could have sent the message; In the latter, the key is held by the recipient, so that only the intended recipient can receive the message accurately.

* + User Authentication: o Passwords: Passwords are the most common form of user authentication. If the user is in possession of the correct password, then they are considered to have identified themselves. In theory separate passwords could be implemented for separate activities, such as reading this file, writing that file, etc. In practice most systems use one password to confirm user identity, and then authorization is based upon that identification.
* Encrypted Passwords: Modern systems do not store passwords in clear-text form, and hence there is no mechanism to look up an existing password. Rather they are encrypted and stored

in that form. When a user enters their password, that too is encrypted, and if the encrypted version match, then user authentication passes.

* One-Time Passwords: One-time passwords resist shoulder surfing and other attacks where an observer is able to capture a password typed in by a user. These are often based on a challenge and a response. Because the challenge is different each time, the old response will not be valid for future challenges. Another option is to have some sort of electronic card with a series of constantly changing numbers, based on the current time. The user enters the current number on the card, which will only be valid for a few seconds. A two-factor authorization also requires a traditional password in addition to the number on the card, so others may not use it if it were ever lost or stolen.
* Biometrics: Biometrics involve a physical characteristic of the user that is not easily forged or duplicated and not likely to be identical between multiple users. Difficulties may arise in the event of colds, injuries, or other physiological changes.

• Firewalling to protect systems and networks: Firewalls are devices (or software) that sit on the border between two security domains and monitor/log activity between them, sometimes restricting the traffic that can pass between them based on certain criteria. For example, a firewall router may allow HTTP: requests to pass through to a web server inside a company domain while not allowing telnet, ssh, or other traffic to pass through. A common architecture is to establish a de-militarized zone, DMZ, which sort of sits "between" the company domain and the outside world, as shown below. Company computers can reach either the DMZ or the outside world, but outside computers can only reach the DMZ. Perhaps most importantly, the DMZ cannot reach any of the other company computers, so even if the DMZ is breached, the attacker cannot get to the rest of the company network. Unfortunately, firewalls have several vulnerabilities:

* Tunneling, which involves encapsulating forbidden traffic inside of packets that are allowed.
* Denial of service attacks addressed at the firewall itself.
* Spoofing, in which an unauthorized host sends packets to the firewall with the return address of an authorized host.

Some specialized forms of firewalls that have been recently developed:

* A personal firewall is a software layer that protects an individual computer. It may be a part of the operating system or a separate software package.
* An application proxy firewall understands the protocols of a particular service and acts as a stand-in for the particular service. For example, and SMTP proxy firewall would accept

SMTP requests from the outside world, examine them for security concerns, and forward only the "safe" ones on to the real SMTP server behind the firewall.

* XML firewalls examine XML packets only, and reject ill-formed packets. Similar firewalls exist for other specific protocols.
* System call firewalls guard the boundary between user mode and system mode, and reject any system calls that violate security policies.[1]

1. **Discuss strategies for implementing the access matrix.**

Strategies for implementing the access matrix:

* + Global Table: The simplest implementation of the access matrix is a global table consisting of a set of ordered triples <domain, object, rights-set>. Whenever an operation M is executed on an object Oj within domain Di, the global table is searched for a triple <Di, Oj, Rk>, with M

 Rk. If this triple is found, the operation is allowed to continue; otherwise, an exception condition is raised. This implementation suffers from several drawbacks. The table is usually large and thus cannot be kept in main memory, so additional I/O is needed. Virtual memory techniques are often used for managing this table. In addition, it is difficult to take advantage of special groupings of objects or domains.

* + Access List for objects: Each column in the access matrix can be implemented as an access list for one object. Obviously, the empty entries can be discarded. The resulting list for each object consists of ordered pairs <domain, rights-set>, which define all domains with a nonempty set of access rights for that object. When an operation M on an object Oj is attempted in domain Di, we search the access list for object Oj, looking for an entry <Di, Rk> with M  Rk. If the entry is found, we allow the operation; if it is not, we check the default set. If M is in the default set, we allow the access. Otherwise, access is denied, and an exception condition occurs. For efficiency, we may check the default set first and then search the access list.
  + Capability list for domains: Rather than associating the columns of the access matrix with the objects as access lists, we can associate each row with its domain. A capability list for a domain is a list of objects together with the operations allowed on those objects. An object is often represented by its physical name or address, called a capability. Simple possession of the capability means that access is allowed. The capability list is associated with a domain, but it is never directly accessible to a process executing in that domain. Rather, the capability

list is itself a protected object, maintained by the operating system and accessed by the user only indirectly. Capability-based protection relies on the fact that the capabilities are never allowed to migrate into any address space directly accessible by a user process. If all capabilities are secure, the object they protect is also secure against unauthorized access. Capabilities were originally proposed as a kind of secure pointer, to meet the need for resource protection that was foreseen as multi programmed computer systems came of age. The idea of an inherently protected pointer provides a foundation for protection that can be extended up to the application level. To provide inherent protection, we must distinguish capabilities from other kinds of objects, and they must be interpreted by an abstract machine on which higher-level programs run.

* + A Lock-Key mechanism: The lock –key scheme is a compromise between access lists and capability lists. Each object has a list of unique bit patterns, called locks. Similarly, each domain has a list of unique bit patterns, called keys. A process executing in a domain can access an object only if that domain has a key that matches one of the locks of the object. As with capability lists, the list of keys for a domain must be managed by the operating system on behalf of the domain. Users are not allowed to examine or modify the list of keys (or locks) directly
  + Comparison: As you might expect, choosing a technique for implementing an access matrix involves various trade-offs. Using a global table is simple; however, the table can be quite large and often cannot take advantage of special groupings of objects or domains. Access lists correspond directly to the needs of users. When a user creates an object, he can specify which domains can access the object, as well as what operations are allowed. However, because access-right information for a particular domain is not localized, determining the set of access rights for each domain is difficult. In addition, every access to the object must be checked, requiring a search of the access list. In a large system with long access lists, this search can be time consuming. Most systems use a combination of access lists and capabilities. When a process first tries to access an object, the access list is searched. If access is denied, an exception condition occurs. Otherwise, a capability is created and attached to the process. Additional references use the capability to demonstrate swiftly that access is allowed. After the last access, the capability is destroyed.[5]

1. **Explain different techniques for Free Space Management under Disk Management.**

Different techniques for free space management:

* Bit Vector: Frequently, the free-space list is implemented as a bit map or bit vector. Each block is represented by 1 bit. If the block is free, the bit is 1; if the block is allocated, the bit is 0. For example, consider a disk where blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, and 27 are free and the rest of the blocks are allocated. The free-space bit map would be 001111001111110001100000011100000 ...

The main advantage of this approach is its relative simplicity and its efficiency in finding the first free block or n consecutive free blocks on the disk. Indeed, many computers supply bitmanipulation instructions that can be used effectively for that purpose. One technique for finding the first free block on a system that uses a bit-vector to allocate disk space is to sequentially check each word in the bit map to see whether that value is not 0, since a 0valued word contains only 0 bits and represents a set of allocated blocks. The first non-0 word is scanned for the first 1 bit, which is the location of the first free block. The calculation of the block number is (number of bits per word) × (number of 0-value words) + offset of first 1 bit. Again, we see hardware features driving software functionality. Unfortunately, bit vectors are inefficient unless the entire vector is kept in main memory (and is written to disk occasionally for recovery needs). Keeping it in main memory is possible for smaller disks but not necessarily for larger ones. A 1.3-GB disk with 512-byte blocks would need a bit map of over 332 KB to track its free blocks, although clustering the blocks in groups of four reduces this number to around 83 KB per disk. A 1-TB disk with 4-KB blocks requires 256 MB to store its bit map. Given that disk size constantly increases, the problem with bit vectors will continue to escalate as well.

* Linked List: Another approach to free-space management is to link together all the free disk blocks, keeping a pointer to the first free block in a special location on the disk and caching it in memory. This first block contains a pointer to the next free disk block, and so on. Recall our earlier example in Bit Vector, in which blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, and 27 were free and the rest of the blocks were allocated. In this situation, we would keep a pointer to block 2 as the first free block. Block 2 would contain a pointer to block 3, which would point to block 4, which would point to block 5, which would point to block 8, and so on (Figure 11.10). This scheme is not efficient; to traverse the list, we must read each block, which requires substantial I/O time. However, traversing the free list is not a frequent action. Usually, the operating system simply needs a free block so that it can allocate that block to a file, so the first block in the free list is used. The FAT method incorporates freeblock accounting into the allocation data structure. No separate method is needed.
* Grouping: A modification of the free-list approach stores the addresses of n free blocks in the first free block. The first n−1 of these blocks are actually free. The last block contains the addresses of another n free blocks, and so on. The addresses of a large number of free blocks can now be found quickly, unlike the situation when the standard linked-list approach is used.
* Counting: Another approach takes advantage of the fact that, generally, several contiguous blocks may be allocated or freed simultaneously, particularly when space is allocated with the contiguous-allocation algorithm or through clustering. Thus, rather than keeping a list of n free disk addresses, we can keep the address of the first free block and the number (n) of free contiguous blocks that follow the first block. Each entry in the free-space list then consists of a disk address and a count. Although each entry requires more space than would a simple disk address, the overall list is shorter, as long as the count is generally greater than 1. Note that this method of tracking free space is similar to the extent method of allocating blocks. These entries can be stored in a balanced tree, rather than a linked list, for efficient lookup, insertion, and deletion.[5]

References:

1. <https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/15_Security.html>
2. [https://www.hysolate.com/learn/sandboxing/understanding-os-security-threats-and-securitycontrols/](https://www.hysolate.com/learn/sandboxing/understanding-os-security-threats-and-security-controls/)
3. [https://en.wikipedia.org//wiki/Stack\_buffer\_overflow](https://en.wikipedia.org/wiki/Stack_buffer_overflow)
4. <https://www.geeksforgeeks.org/types-of-virus/>
5. Operating system concepts sixth edition