Unit 7: Introduction to Operating Systems

1. Describe mitigations to make operating systems more secure.

In computer systems, safeguarding against unauthorised access and ensuring that each process adheres to established policies is crucial. The protection system must be adaptable enough to enforce various policies set by system managers or individual users. Protection mechanisms are also vital for application programmers to defend resources against misuse. According to Abraham et al. (2018), the principle of least privilege is essential in designing an operating system as it allocates users, programs, and procedures with the necessary permissions to perform their tasks. This prevents malicious code from obtaining root privileges; defined permissions can block harmful operations. Compartmentalisation is another critical principle that protects individual system components through specific access restrictions. Careful use of these restrictions can improve system security and produce an audit trail to track unauthorised accesses. However, no single principle can ensure security vulnerabilities, and a defence-in-depth approach should be employed, applying multiple layers of protection on top of one another.

Saltzer & Schroeder (1975) outlined the three security violations: unauthorised information release, unauthorised information modification, and unauthorised denial of use. Security techniques such as labelling files with authorised user lists, password verification, and controlling system changes are implemented to prevent these violations. Authentication techniques are used to verify the identity of a person requesting access to a computer system. Protection schemes vary in functional properties, such as controlled sharing and user-programmed sharing controls. A secure system's ultimate goal is to prevent unauthorised use of information.

According to Rouse (2023), it is crucial to mitigate security risks in operating systems to protect against various threats and vulnerabilities. Employing a range of techniques and strategies can safeguard the integrity of an operating system, preventing potential dangers and breaches. This allows applications and programs to function without unauthorised interference.

To ensure the security of an OS, several methods can be employed, including regularly updating the operating system with patches, installing up-to-date antivirus software and engines, scrutinising all incoming and outgoing network traffic with a firewall, and creating secure user accounts with appropriate privileges only (Rouse, 2023).

Protection and security mechanisms ensure that data and programs are used only by authorised users and in a desired manner. Passwords are a principal security tool, and encryption upholds confidentiality. Operating systems employ protection measures to specify file access privileges and ensure strict adherence. Security measures guard against interference from nonusers, such as Trojan horses, viruses, worms, and buffer overflows. Access control lists, capability lists, protection domains control file sharing, and security classifications reflect a system's ability to withstand threats (GeeksforGeeks, 2023).

2. **Explain some of the critical risks and vulnerabilities associated with operating systems.**

Operating System vulnerabilities can be caused by development errors or outdated software. Attackers exploit these vulnerabilities to gain easy access to a system and execute attacks like Ransom denial of service. Vulnerabilities reported by databases

and vendors include code execution, overflow, exploits, memory corruption, and others (Softić & Vejzović, 2022).

Maintaining computer systems, data security, and integrity is paramount (Abraham et al., 2018). Therefore, ensuring the safety of computer systems, primarily financial and corporate data, requires significant efforts. Security breaches can significantly disrupt the operation of a corporation. Security violations can be intentional or accidental, but it is easier to prevent accidental misuse. Breaches of confidentiality, integrity, and availability are widespread, and attackers use standard methods such as masquerading, replay attacks, and man-in-the-middle attacks to breach security.

Furthermore, Abraham et al. (2018) added that attackers can escalate their privileges to gain more access to a system, and detecting and preventing all privilege escalation attacks is challenging. Achieving complete protection is impossible, but most intruders can be deterred by making the cost of breaching security high. To protect a system, four levels of security measures are necessary: physical, network, operating system, and application. These measures include securing the physical site, network, and operating system and securing third-party applications.

The four-layer security model is like a chain, where any weakness can result in system compromise. Authorisation must be cautious as even authorised users can be malicious or deceived through social engineering. Phishing is a frequent type of social engineering attack. Hardware protection features are essential for an overall protection scheme. As security countermeasures become more sophisticated, intruders become more advanced, necessitating additional security tools to block their techniques.

In addition, operating systems are a crucial component of any computing environment, providing a platform for running software and managing hardware resources. However,

due to their complexity and widespread use, attackers often target operating systems, looking for vulnerabilities and weaknesses to exploit (Abraham et al. 2018).

However, IT professionals must be aware of several critical risks and vulnerabilities associated with operating systems. One of the most common vulnerabilities is software vulnerabilities, which include buffer overflows and privilege escalation. Buffer overflows occur when an attacker overwrites a program's memory, potentially leading to the execution of arbitrary code. Privilege escalation involves exploiting vulnerabilities to gain higher-level access privileges than initially granted (Haber, 2023).

Haber (2023) also pointed out that malware and malicious software are other significant risks associated with operating systems. Viruses, trojans, and worms are malware that can wreak havoc on systems, steal data, and compromise security.

Little known to many software vendors, zero-day exploits are a category of software vulnerability that can be used by cybercriminals and state-sponsored groups to access networks and extract sensitive data illegally. To combat these threats, organisations should prioritise activities such as patching vulnerabilities, implementing more comprehensive mitigations, and encouraging transparency and cooperation between security defenders and software vendors (Poireault, 2023).

As identified by Aztechit (2020), insufficient patching and updates can also leave operating systems vulnerable to known exploits.

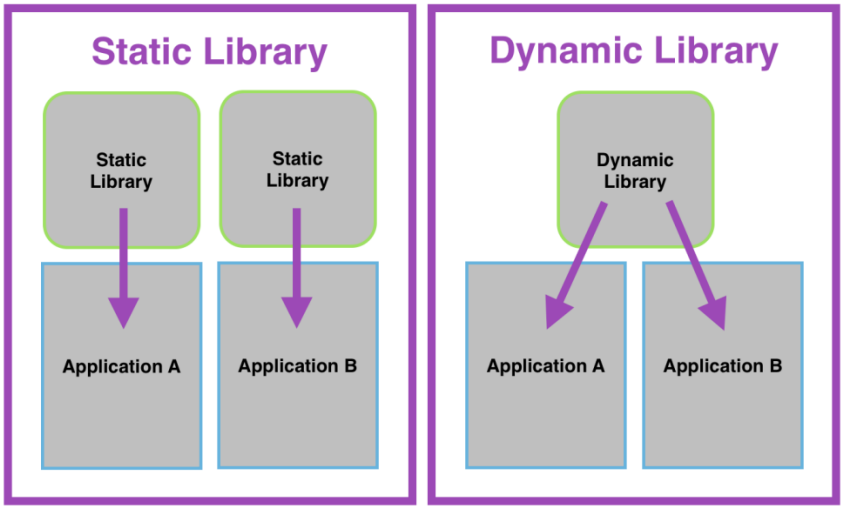
To mitigate these risks and vulnerabilities, organisations must adopt a proactive approach to security, which includes regular patching, security audits, employee training, intrusion detection, and protection best practices (Sachdeva, 2023).

3. **Outline the differences between static and dynamic libraries and when to use each.**

A library is a valuable resource for programmers, comprising a collection of code that other programs can use. It consists of one or more header files that offer crucial information about variables, functions, classes, and other programming elements supplied by the library. The library's actual code implementation is found in a precompiled binary file that is distributed along with the headers (Triangles, 2020).

Triangles (2020) explains that the header file is incorporated into the program's source code to employ the library, which informs the compiler of the library's contents. During the compilation process, the binary file is linked to the final executable, ensuring seamless integration.

Compilers provide libraries to link object files together. Libraries contain multiple object files, making it faster to find symbols during linking. Unix systems have two library types: static and shared. Static libraries are linked during compilation and not relevant during runtime. Shared libraries are linked during compile time but loaded by a dynamic loader during runtime. Shared libraries save memory and disk space, but re-compiling them can cause issues (Fernández, 2022).



(Fernández, 2022)

According to Fernández (2022), dynamic libraries enable multiple applications to access the same file without duplication, but a corrupted library can affect the functionality of the executable file. They are smaller in size, and modifications do not necessitate recompilation. Conversely, static libraries are embedded inside the executable file and execute faster, making them less susceptible to vulnerabilities. However, updates require relinking and recompilation; the resulting file is typically larger.

Kumar (N.D.) wrote that the differences between static and dynamic linking are significant. Static linking involves copying all library modules used in the program into the final executable image, resulting in larger file sizes. On the other hand, dynamic linking only includes the names of the external libraries in the final executable file, reducing the size of executable programs and saving memory and disk space. Additionally, dynamic linking allows individual shared modules to be updated and recompiled, providing a significant advantage over static linking. However, dynamically linked programs depend on a compatible library and may encounter compatibility issues if a library is changed or removed. It is essential to consider these differences when choosing between static and dynamic linking.

Dynamic linking enables the loader to resolve external symbols in user code during load time, resulting in smaller executable programs. Nevertheless, this comes at a cost of reduced locality of reference and performance trade-off. On the other hand, static linking consolidates all code in a single executable module, leading to more efficient library references and increasing the file and memory size (IBM, 2023).

References:

Abraham, S., Greg, G. & Peter Baer, G. (2018). *Operating System Concepts.* 10th ed. Wiley.

Saltzer, J.H. & Schroeder, M.D. (1975). The protection of information in computer systems. *Proceedings of the IEEE*, 63(9), pp.1278–1308. doi:https://doi.org/10.1109/proc.1975.9939.

Rouse, M. (2019). *What is Operating System Security (OS Security)? - Definition from Techopedia*. [online] Available at: https://www.techopedia.com/definition/24774/operating-system-security-os-security.

GeeksforGeeks. (2023). *Operating System Security*. [online] Available at: https://www.geeksforgeeks.org/operating-system-security/.

Softić, J. & Vejzović, V. (2022). *Operating Systems Vulnerability - An Examination of Windows 10, macOS, and Ubuntu from 2015 to 2021*. [online] Available at: https://www.astesj.com/v07/i06/p25/ [Accessed 28 Sep. 2023].

Haber, M. (2021). *Privilege Escalation Attack & Defense Explained | BeyondTrust*. [online] www.beyondtrust.com. Available at: https://www.beyondtrust.com/blog/entry/privilege-escalation-attack-defense-explained.

Poireault, K. (2023). *A Guide to Zero-Day Vulnerabilities and Exploits for the Uninitiated*. [online] Infosecurity Magazine. Available at: https://www.infosecurity-magazine.com/news-features/guide-zero-day-vulnerabilities/

Aztechit (2020). *Cyber Security Threats and Vulnerabilities 101* | *Definition & Types*. [online] Available at: https://www.aztechit.co.uk/blog/cyber-security-threats-and-vulnerabilities.

Symantec (2015). *INTERNET SECURITY THREAT REPORT.* Available from: https://scadahacker.com/library/Documents/Threat_Intelligence/Symantec%20-%20Internet%20Security%20Threat%20Report%20-%202015.pdf

Sachdeva, D. (2023). *Large Organizations, High Security-Risks: The Expanding Attack Surface in the Digital Age"*. [online] Available at: https://kratikal.com/blog/large-organizations-high-security-risks-the-expanding-attack-surface-in-the-digital-age/

Tringles (2020). *A journey across static and dynamic libraries*. [online] Internal Pointers. Available at: https://www.internalpointers.com/post/journey-across-static-dynamic-libraries

Fernández, T. (2022). *Static library vs Dynamic library*. [online] Available at: https://www.linkedin.com/pulse/static-library-vs-dynamic-tatiana-fern%C3%A1ndez/

Kumar, K. (N.D.). *Difference Between Static and Dynamic Library Linking*. [online] cs-fundamentals.com. Available at: https://cs-fundamentals.com/tech-interview/c/difference-between-static-and-dynamic-linking.

IBM (2023). *When to use dynamic linking and static linking*. [online] Available at: https://www.ibm.com/docs/en/aix/7.2?topic=techniques-when-use-dynamic-linking-static-linking