UNIT 3: TEAM Activity: Factors Determining Whether a Programming Language is Secure

Team Discussion: What is a Secure Programming Language?

According to Pillai (2017), key factors that determine the security of a programming language include:

1. Memory Safety: Secure languages provide mechanisms to manage memory without exposing the programmer to risks of buffer overflows or memory corruption. For instance, languages that handle memory management automatically (e.g., garbage collection) are less prone to memory safety issues.

2. Type Safety: Languages with strong type systems prevent operations that could result in type-related vulnerabilities. This includes enforcing strict type checking to avoid type errors and unexpected behavior.

3. Error Handling: Effective error handling mechanisms help manage and contain faults, reducing the likelihood of vulnerabilities due to unhandled exceptions or errors.

4. Access Control: Secure languages implement robust access control measures to prevent unauthorised access to system resources and sensitive data.

5. Standard Libraries: Secure languages provide libraries and frameworks that are designed with security in mind, avoiding common pitfalls and vulnerabilities.

Cifuentes & Bierman (2019) likely emphasise similar principles, particularly focusing on how language design can prevent common security issues through features like type safety and error handling.

Python as a Secure Language

Pillai (2017) discusses Python’s security features:

- Memory Safety: Python’s automatic memory management and garbage collection help prevent common issues like buffer overflows and memory leaks, enhancing its security.
- Type Safety: Although Python uses dynamic typing, which introduces some risks, its type system helps avoid type-related errors during runtime.
- Error Handling: Python provides robust error handling with exceptions, which can catch and manage errors effectively, reducing the risk of crashes and vulnerabilities.

However, Pillai (2017) might also note that Python's security can be affected by its dynamic nature and reliance on external libraries, which could introduce vulnerabilities.

Python vs. C for Operating Systems

Cifuentes & Bierman (2019) and Pillai (2017) address the suitability of languages for system-level programming:

- Performance: C is favoured for operating systems due to its low-level access and performance efficiency. It allows precise control over system resources and memory, which is crucial for OS development.
- Memory Management: C offers direct memory management capabilities, which are essential for creating efficient and high-performance operating systems. Python’s automatic memory management is less suitable for this level of control.
- Access to Hardware: C provides direct hardware access and system-level operations, which are necessary for OS functionality. Python’s abstraction layers and lack of low-level access make it less appropriate for such tasks.

In summary, while Python offers good security features for general applications, C's low-level control and performance make it more suitable for operating system development.

Analyzing Secure Programming Languages: Insights from Pillai and Cifuentes & Bierman

The security of a programming language is determined by several critical factors. According to Pillai (2017), memory safety is paramount; languages that manage memory automatically, such as those with garbage collection, minimize risks related to buffer overflows and memory leaks. Additionally, type safety plays a crucial role. Secure languages enforce strict type checking, preventing operations that could lead to type errors and vulnerabilities (Cifuentes, 2019). Effective error handling mechanisms are also essential, as they manage exceptions and errors to reduce the risk of security breaches. Access control features further enhance security by safeguarding against unauthorized resource access (Pillai, 2017). Finally, secure programming languages offer robust standard libraries designed to avoid common vulnerabilities.

In terms of security, Pillai (2017) highlights Python's strengths. Python's automatic memory management and garbage collection mitigate risks associated with manual memory handling. Although Python employs dynamic typing, which introduces some vulnerabilities, its type system still helps in managing type-related errors (Pillai, 2017). Moreover, Python's exception handling framework allows effective management of runtime errors, enhancing overall security (Pillar, 2017). However, Python's reliance on external libraries and its dynamic nature can introduce potential security risks that need to be managed.

Comparatively, when evaluating Python versus C for operating system development, Cifuentes & Bierman (2019) and Pillai (2017) suggest that C is more suitable. C's low-level control and efficiency are crucial for system programming, offering precise management of system resources and memory. Its direct access to hardware and system resources is essential for developing high-

performance operating systems. Conversely, Python's automatic memory management and high-level abstractions, while beneficial for application development, lack the control required for effective operating system design (Cifuentes, 2019). Thus, while Python excels in security features for general applications, C remains the preferred choice for system-level programming due to its performance and control.

References:

Cifuentes, C. & Bierman, G. (2019) *What is a Secure Programming Language?* 3rd Summit on Advances in Programming Languages (SNAPL).136(3): 1 - 15.

Pillai, A.B. (2017) *Software Architecture with Python*. Birmingham, UK. Packt Publishing Ltd.