**Discussion: What is a Secure Programming Language?**

1. **What factors determine whether a programming language is secure or not?**

According to Cifuentes & Bierman (2019), the ability to address buffer overflow errors, injection faults due to unvalidated or improperly validated inputs (Pillai, 2017), and information leak errors determine whether a programming language is secure or not. Besides buffer overflows, Pillai (2017) also mentions the importance of considering vulnerable arithmetic or integer overflow errors too. Pillai (2017) provides clear examples of information leaks, such as server meta information, open index pages, open ports, race conditions, etc.

1. **Could Python be classed as a secure language? Justify your answer.**

It depends on how Python programming is leveraged, as Pillai (2017) points out. No mainstream programming language, including Python, offers a general safe abstraction for injection errors (Cifuentes & Bierman, 2019). Nevertheless, via the ‘taint tracking’ library, Python can provide taint tracking capabilities to form the basis of a more general abstraction to address injection errors. Moreover, there is no language-level feature in any mainstream languages, including Python, to address vulnerabilities resulting from information leak errors. However, tracking or data shadowing of sensitive data and a DSL for data-centric applications can be leveraged to address such vulnerabilities (Cifuentes & Bierman, 2019).

The ‘eval’ function in Python may also be a security vulnerability, as it can allow passing arbitrary strings to it, which can result in evaluating potentially dangerous codes (Pillai, 2017). Pillai (2017) points out that integer overflow errors could occur in Python 2 via the ‘xrange’ and ‘range’ functions for instance, as they used plain integer objects (of type ‘int’) instead of converting them to the ‘long’ type, which is solely limited by the system’s memory. Nevertheless, in Python 3, this issue is solved by unifying the data types ‘int’ and ‘long’ into one ‘int’ type, with the ‘range’ objects managing memory internally (Pillai, 2017).

1. **Python would be a better language to create operating systems than C. Discuss.**

C is a better language for creating operating systems (OS) than Python, despite the creation of OS in Python is not impossible, considering the ability of Python to provide wrappers to the system calls to support event-driven programs via the ‘select’ module (Pillai, 2017). Although Python is more memory-efficient than C, to create operating systems, the performance overhead of Python’s managed memory runtime is still not feasible (Cifuentes & Bierman, 2019). Moreover, Python is a high-level interpreted programming language, requiring an interpreter to execute Python codes. C is a low-level compiled programming language, thus being faster than Python, that enables the creation of standalone executables, differently from Python which needs a runtime to execute its codes (Tanenbaum, 2009). Python does not enable reading or writing to low-level computing resources, such as a CPU or memory; instead, such resources can be accessed via C (Tanenbaum, 2009). Furthermore, by using Python, a bootloader cannot be written either.

**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.

* Chapter 2.
* Chapter 6.
* Chapter 7.
* Chapter 8.

Tanenbaum, A. (2009) *Modern operating systems*. Pearson Education, Inc.