Team Discussion: What is a Secure Programming Language?

You should read Chapters 2,6,7,8 of the course text (Pillai, 2017) and Cifuentes & Bierman (2019) and then answer the questions below, adding them as evidence to your e-portfolio.

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

Secure coding is creating software from the ground up with security in mind. This entails early security requirement identification, threat modelling to foresee potential risks, and the application of secure coding techniques. As stated by Pillai (2017), significant duties include identifying critical assets, assessing software design, analysing implementation details, confirming logic and syntax, doing unit and Blackbox testing, and addressing any security flaws found.

According to Cifuentes & Bierman (2019), there are three principal vulnerabilities:

1.1 **Buffer Errors:** Buffer overflow attacks overload memory storage and can corrupt memory beyond the buffer's boundaries. The attack can be stack-based or heap-based.

1.2 **Injection Errors:** Injection errors have several vulnerabilities, including XSS, SQL injection, code injection, and OS command injection. XSS allows attackers to inject malicious code into websites, compromising data confidentiality and integrity. Websites are vulnerable if they display unsanitised user-supplied data. There are three types of XSS: reflected, stored, and DOM-based.

1.3 **Information Leak Errors:** Information leakage releases information to an untrusted environment. It can occur through various means, including log files, caching, and error messages. The most common type is through log files.

The wrong abstractions are used in programming languages, which results in vulnerable code. Information leaks come from manually tracking sensitive data, whereas buffer and injection mistakes are caused by manual pointer and string management, respectively.

Any secure programming language deserving of the designation must have first-class support for each of the three categories, according to Cifuentes & Bierman's (2019) thesis. Consequently, a secure programming language must provide best-in-class linguistic support, including memory security, data integrity, and secrecy methods to solve these issues.

Developers need safe abstractions to write secure code rather than migrating millions of lines of insecure code.

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

Python's syntax is simple, readable, and has a clear way of doing things. Moreover, it boasts a well-tested set of standard library modules, indicating that it is secure (Pillai, 2017).

However, Pillai (2017) states that security concerns related to Python encompass various aspects, such as evaluating expressions, reading input, overflow errors, and serialisation issues. Web applications created using frameworks like Django and Flask are particularly vulnerable. To mitigate these risks, it is imperative to employ secure coding practices such as avoiding pickle exec and ensuring integer overflow protection. Additionally, it is crucial to adopt safer input methods.

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

An operating system (OS) manages application programs and provides services to users through a command-line interface (CLI) or graphical user interface (GUI) (Bigelow, 2021).

It needs a solid foundation in computer science fundamentals, basic programming, and high-level and low-level programming languages to develop an operating system. Assembly languages directly communicate with the CPU, while x86 architecture and C programming are commonly used for OS development (saijyosthanakanchi555, 2021).

Dougvj (2012) explains that Python is a high-level programming language that requires an abstraction layer, like the Kernel, to access hardware and perform low-level data structure manipulation. It can create an operating system with only the low-level components written in C and assembly and the majority written in Python. On the one hand, writing an OS in assembly code is feasible but time-consuming and skill-intensive. OS development is made simpler by high-level languages like C and C++, with C being beneficial as a middle-level language with both high-level constructs and low-level features. C was created with system-level and embedded program development in mind. It is a highly portable structured language that enables the division of extensive programs into more straightforward functions, making it suitable for scripting system applications on Windows, UNIX, and Linux (Dahanayaka, 2021).

Operating systems are typically written in C, a portable and powerful programming language that can create complex designs. While assembly language is used for some parts of the Windows kernel, C is used for the seed as a whole. Other programming languages like Python are also options, but C remains the best choice for system programming due to its low and high-level operations (Lemp.io, 2023).

Formative activities: Collaborative discussion in Unit 1 – Summary Post

My previous post discussed Cross-Site Scripting (XSS) and its potential security risks. XSS is a type of vulnerability that occurs when malicious scripts are inserted into a trusted website or app, enabling hackers to steal data, take control of accounts, and execute harmful code on the user's browser (OWASP, N.D.). This issue arises when applications fail to validate, filter, or sanitise user-supplied data. To illustrate the weaknesses of XSS, I created a Sequence Diagram. However, my tutor suggested using an Activity Diagram, which visually represents a system's actions or control flow more effectively (Smartdraw, N.D.). I agree and have included an alternative activity diagram to demonstrate a malicious XSS attack based on the ideas of Gupta & Sharma (2012).

During a recent discussion, my colleague Sebastian raised an essential point regarding XSS vulnerabilities. Selvamani et al. (2010) found that careless use of the Document Object Model in JavaScript can create vulnerabilities where malicious code from another page can alter the principle of the first page on a local system. It is essential to understand these distinctions, as each one requires specific countermeasures. Sebastian ended his point with a relevant question: How can developers protect their applications against various XSS attacks? IBM (N.D.) suggests that to prevent XSS attacks; the application must validate all input data, only allow listed data, and encode all variable output on pages before returning it to the user.

I have commented on my colleagues' posts regarding cryptographic flaws and insecure design and implementation of APIs.

After analysing colleague Adesola's post on Cryptographic Flaws, I realised the importance of web application security. Further research revealed potential implications such as data breaches, loss of trust, legal and compliance issues, and intellectual property theft, as explained in Ali's research (2023).

My colleague Sebastien provided insightful information that helped me understand the importance of security in API projects. I learned more about RESTful, BOLA, and UBER API security incidents, illustrating the need for in-API security.

References:

Pillai, A.B. (2017). *Software architecture with Python*. Packt Publishing Ltd.

Cifuentes, C. & Bierman, G. (2019). What is a secure programming language?. In *3rd Summit on Advances in Programming Languages (SNAPL 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Bigelow, S. (2021). *What is an Operating System (OS)? Definition, Types and Examples*. [online] WhatIs.com. Available at: https://www.techtarget.com/whatis/definition/operating-system-OS.

saijyosthanakanchi555 (2021). *Guide to Build an Operating System From Scratch*. [online] Available at: https://www.geeksforgeeks.org/guide-to-build-an-operating-system-from-scratch/.

Dougvj (2012). *Is it possible to create an operating system using Python?* [online] Available at: https://stackoverflow.com/questions/10904721/is-it-possible-to-create-an-operating-system-using-python [Accessed 26 Aug. 2023].

Dahanayaka, M. (2021). *Create Your Own Operating System*. [online] Medium. Available at: https://medium.com/@mekaladahanayaka80/create-your-own-operating-system-a4b1c179c28f#:~:text=Using%20a%20high%2Dlevel%20language [Accessed 27 Aug. 2023].

Lemp.io. (2023). *HOW TO BUILD AN OPERATING SYSTEM IN C.* Available at: https://lemp.io/how-to-build-operating-system-in-c/ [Accessed 27 Aug. 2023].

OWASP (N.D.). *Cross-Site Scripting (XSS) Software Attack | OWASP Foundation*. [online] Available at: https://owasp.org/www-community/attacks/xss/#:~:text=Cross%2DSite%20Scripting%20(XSS).

Smartdraw (N.D.). *Activity Diagram - Activity Diagram Symbols, Examples, and More*. [online] Available at: https://www.smartdraw.com/activity-diagram/#:~:text=Learn%20More-.

Gupta, S. & Sharma, L. (2012). *Exploitation of Cross-Site Scripting (XSS) Vulnerability on Real World Web Applications and its Defense.* Available at: https://research.ijcaonline.org/volume60/number14/pxc3883594.pdf

Selvamani, K., Duraisamy, A. & Kannan, A. (2010). *Protection of Web Applications from Cross-Site Scripting Attacks in Browser Side*. [online] arXiv.org. doi:https://doi.org/10.48550/arXiv.1004.1769.

IBM (N.D.). *Protect from cross-site scripting attacks*. Available at: https://www.ibm.com/garage/method/practices/code/protect-from-cross-site-scripting/.

Ali, Z. (2023). *Cryptographic Failures: Understanding the Pitfalls and Impact.* [online] Available at: https://www.linkedin.com/pulse/cryptographic-failures-understanding-pitfalls-impact-zahid-ali/.

Codio Activity - Exploring Python tools and features

Part I

In this example, you will compile and run a program in C using the **Codio workspace** provided (Buffer Overflow in C). The program is already provided as bufoverflow.c - a simple program that creates a buffer, asks you for a name and prints it back to the screen.

This is the code in bufoverflow.c (also available in the Codio workspace):

```
#include <stdio.h>
int main(int argc, char **argv)
{
char buf[8]; // buffer for eight characters
printf("enter name:");
gets(buf); // read from stdio (sensitive function!)
printf("%s\n", buf); // print out data stored in buf
return 0; // 0 as return value
{
```

Now, compile and run the code. To test it, enter your first name (or at least the first 8 characters of it); you should get the output of just your name repeated back to you.

Run the code a second time (from the command window; this can be achieved by entering ./bufoverflow on the command line). This time, enter a string of 10 or more characters.

1. What happens?
2. What does the output message mean?

Output after running the code:

```
codio@augustjustice-sharonstamp:~/workspace$ gcc bufoverflow.c -o bufoverflow && ./bufoverflow
bufoverflow.c: In function ‘main’:
bufoverflow.c:8:5: warning: implicit declaration of function ‘gets’; did you mean ‘fgets’? [-Wimplicit-function-declaration]
    gets(buf);           // read from stdio (sensitive function!)
    ^~~~
    fgets
/tmp/ccnxtD3J.o: In function `main':
```

```
bufoverflow.c:(.text+0x3c): warning: the `gets' function is dangerous and should not be used.
Enter name: Hainadin
Hainadin

codio@augustjustice-sharonstamp:~/workspace$ gcc bufoverflow.c -o bufoverflow && ./bufoverflow
bufoverflow.c: In function ‘main’:
bufoverflow.c:8:5: warning: implicit declaration of function ‘gets’; did you mean ‘fgets’? [-Wimplicit-
function-declaration]
    gets(buf);              // read from stdio (sensitive function!)
    ^~~~
    fgets
/tmp/ccQjNn6c.o: In function `main':
bufoverflow.c:(.text+0x3c): warning: the `gets' function is dangerous and should not be used.
Enter name: HainadineC
HainadineC
*** stack smashing detected ***: <unknown> terminated
Aborted (core dumped)
```

Some functions are dangerous as they lack security measures. Examples include gets() and the >> operator, both of which do not perform bounds checking and can easily be exploited by attackers to overflow the destination buffer (CWE, N.D.). For instance, Sharma (2022) wrote that the gets() function in C solves this issue by accepting a size limit and preventing overflows.

When data is written to a buffer that is too small to hold it, a buffer overflow occurs. This overwrites adjacent memory addresses and can be avoided by implementing proper bounds checking (Cobb, 2021). Buffer overflows can corrupt a program's memory, resulting in unpredictable behaviour or program crashes. The operating system's memory protection mechanisms detect these violations and terminate the program to prevent security risks and data corruption (Welekwe, 2020).

The compiler generates a "stack smashing detected" error to defend against buffer overflows caused by input exceeding buffer capacity. C code with a buffer capacity of eight can cause this error if exceeded (Educative, N.D.).

1. **What happens?**

Providing a string of 10 or more characters to the code will cause a buffer overflow. The `buf` array is only 8 characters, and the `gets()` function does not check the input size, leading to memory overwriting. This can cause crashes, unintended behaviour, or exploitation.

2. **What does the output message mean?**

The output message is a fault error. This happened because the buffer overflow corrupts the program's memory, leading to unpredictable behaviour. The operating system's memory protection mechanisms detect this violation and terminate the program to prevent potential security risks or data corruption.

Part II

Now, carry out a comparison of this code with one in Python (Buffer Overflow in Python), following these instructions:

In the Codio workspace, you will be using the file called Overflow.py:

```
buffer=[None]*10
for i in range (0,11):
   buffer[i]=7
print(buffer)
```

Run your code using Python overflow.py (or use the codio rocket icon).

1. What is the result?

- Read about Pylint at http://pylint.pycqa.org/en/latest/tutorial.html
- Install pylint using the following commands:
  pip install pylint (in the command shell/ interpreter)
- Run pylint on your Overflow.py file and evaluate the output:
  pylint Overflow.py

- (Make sure you are in the directory where your file is located before running Pylint)

2. What is the result? Does this tell you how to fix the error above?

Output after running the code:

```
codio@gridbishop-specialnumber:~/workspace$ python3 Overflow.py
Traceback (most recent call last):
  File "Overflow.py", line 3, in <module>
    buffer[i]=7
IndexError: list assignment index out of range
```

1. Running the Python code results in an " IndexError: list assignment index out of range". This is because, in the code, the buffer has 10 elements, but the loop attempts to write through 15 elements, which results in an error. When the count reaches 10, trying to assign buffer[10] = 7 will result in an "IndexError" since index 10 is out of bounds for the buffer list.

2. **Output after running the code:**

```
pylint Overflow.py
```

```
codio@gridbishop-specialnumber:~/workspace$ pylint Overflow.py
************* Module Overflow
Overflow.py:4:0: C0303: Trailing whitespace (trailing-whitespace)
Overflow.py:5:0: C0304: Final newline missing (missing-final-newline)
Overflow.py:1:0: C0103: Module name "Overflow" doesn't conform to snake_case naming style
(invalid-name)
Overflow.py:1:0: C0114: Missing module docstring (missing-module-docstring)

------------------------------------------------------------------
Your code has been rated at 0.00/10 (previous run: 0.00/10, +0.00)
```

I have found four errors in the code. To retrieve the solution for each error, I can run the following command according to the explanation of Pylint 3.0.0a8-dev0 documentation (N.D.): "pylint --help-msg=" and add the message information between parentheses.

```
codio@gridbishop-specialnumber:~/workspace$ pylint --help-msg=missing-module-docstring
:missing-module-docstring (C0114): *Missing module docstring*
```

```
  Used when a module has no docstring. Empty modules do not require a docstring.
  This message belongs to the basic checker.

codio@gridbishop-specialnumber:~/workspace$ pylint --help-msg=trailing-whitespace
:trailing-whitespace (C0303): *Trailing whitespace*
  Used when there is whitespace between the end of a line and the newline. This
  message belongs to the format checker.

codio@gridbishop-specialnumber:~/workspace$ pylint --help-msg=trailing-whitespace
:trailing-whitespace (C0303): *Trailing whitespace*
  Used when there is whitespace between the end of a line and the newline. This
  message belongs to the format checker.

codio@gridbishop-specialnumber:~/workspace$ pylint --help-msg=invalid-name
:invalid-name (C0103): *%s name "%s" doesn't conform to %s*
  Used when the name doesn't conform to naming rules associated to its type
  (constant, variable, class...). This message belongs to the basic checker.
```

Codio Activity: The Producer-Consumer Mechanism

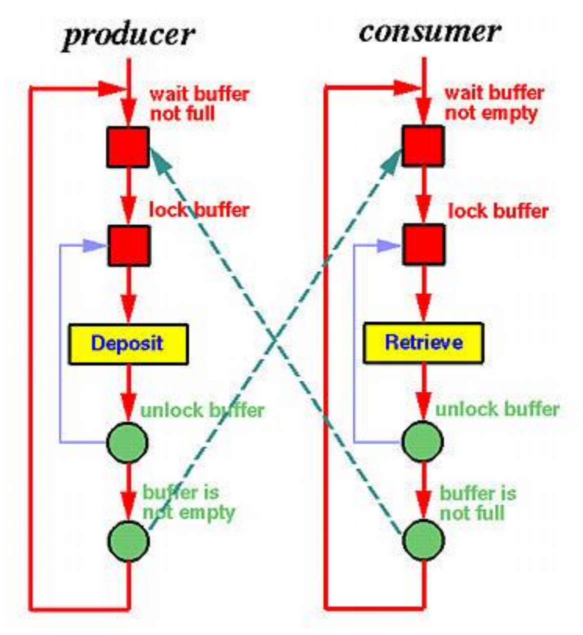
Producer/Consumer Problem (also known as the ‘bounded buffer’ problem):

- A ‘producer’ produces items at a particular (unknown and sometimes unpredictable) rate.
- A ‘consumer’ is consuming the items – again, at some rate.

For example, a producer-consumer scenario models an application producing a listing that a printer process must consume. A keyboard handler creates a data line that an application program will consume. This is shown in the picture below (Shene, 2014).

Items are placed in a buffer when produced, so:

- Consumers should wait if there isn’t an item to consume
- Producer shouldn’t ‘overwrite’ an item in the buffer



Synchronisation is necessary because:

- If the consumer has not taken out the current value in the buffer, then the producer should not replace it with another.
- Similarly, the consumer should not consume the same value twice.

Task

Run producer-consumer.py in the provided Codio workspace (**Producer-Consumer Mechanism**), where the queue data structure is used.

A copy of the code is available here for you.

# code source: https://techmonger.github.io/55/producer-consumer-python/

```
from threading import Thread
from queue import Queue

q = Queue()
final_results = []

def producer():
    for i in range(100):
        q.put(i)


def consumer():
    while True:
        number = q.get()
        result = (number, number**2)
        final_results.append(result)
        q.task_done()


for i in range(5):
   t = Thread(target=consumer)
   t.daemon = True
   t.start()

producer()

q.join()

print (final_results)
```

Answer the following questions:

1. How is the queue data structure used to achieve the purpose of the code?
2. What is the purpose of q.put(I)?
3. What is achieved by q.get()?
4. What functionality is provided by q.join()?
5. Extend this producer-consumer code to make the producer-consumer scenario available securely. What technique(s) would be appropriate to apply?

1. **How is the queue data structure used to achieve the purpose of the code?**

Using the queue data structure enables the implementation of the producer-consumer mechanism (Tech Monger, N.D). In this case, the consumer thread retrieves items from the queue, which acts as a buffer for the producer's goods until they are ready for use by the customer.

2. **What is the purpose of q.put(I)?**

In this programme, the producer adds items (numbers ranging from 0 to 99) to the queue using the q.put(i) method. A new item is added to the queue on each iteration of the producer loop to imitate the production of data that needs to be processed.

3. **What is achieved by q.get()?**

The consumer threads access entries from the queue using q.get(). Every consumer thread in this code reliably takes items from the queue, squares the result, and stores the result in the list labelled "final_results."

4. **What functionality is provided by q.join()?**

The q.join() function waits for all items in the queue to be handled and designated as complete by q.task_done(), ensuring that the program won't progress until all items are processed.

5. **Extend this producer-consumer code to make the producer-consumer scenario available securely. What technique(s) would be appropriate to apply?**

Thread synchronisation prevents multiple processes or threads from simultaneously executing the same critical section. Synchronisation techniques control access to the area, avoiding race conditions and unpredictable variable values (GeeksforGeeks, 2017).

Employing thread synchronisation and locking techniques guarantees the threads' safety when numerous threads deal with shared resources, such as a queue. To ensure safe access to shared data, mutexes (locks) can be implemented (GeeksforGeeks, 2022). The following code has been updated to increase security based on concepts from Python Tutorial (N.D.):

```
from threading import Thread, Lock
from queue import Queue

q = Queue()
final_results = []
lock = Lock()  # Create a lock for shared data protection

def producer():
   for i in range(100):
       q.put(i)

def consumer():
   while True:
       number = q.get()
       result = (number, number**2)
```

```
        with lock:  # Acquire the lock before modifying shared data
            final_results.append(result)

        q.task_done()

for i in range(5):
    t = Thread(target=consumer)
    t.daemon = True
    t.start()

producer()

q.join()

print(final_results)
```

To ensure secure access to the "final_results" list, a lock has been added to the code that consumer threads use to append results. This "lock" prevents unauthorised access. By using the "with lock" statement, potential race conditions are avoided, and thread safety is promoted. This limits access to shared data to one thread at a time.

References:

Sharma, T. (2022). *gets() Function in C*. [online] Scaler Topics. Available at: https://www.scaler.com/topics/gets-in-c/ [Accessed 27 Aug. 2023].

CWE (N.D.). *CWE - CWE-242: Use of Inherently Dangerous Function (4.12)*. [online] Available at: https://cwe.mitre.org/data/definitions/242.html#:~:text=The%20gets()%20function%20is [Accessed 27 Aug. 2023].

Cobb, M. (2021). *What is a Buffer Overflow? How Do These Types of Attacks Work?* [online] SearchSecurity. Available at: https://www.techtarget.com/searchsecurity/definition/buffer-overflow.

Educative (N.D.). *What is the 'stack smashing detected' error?* [online] Available at: https://www.educative.io/answers/what-is-the-stack-smashing-detected-error [Accessed 26 Aug. 2023].

Welekwe, A. (2020). *Buffer overflow vulnerabilities and attacks explained*. [online] Comparitech. Available at: https://www.comparitech.com/blog/information-security/buffer-overflow-attacks-vulnerabilities/.

Tech Monger (N.D). *Producer Consumer Model in Python - Tech Monger*. [online] Available at: https://techmonger.github.io/55/producer-consumer-python/ [Accessed 30 Aug. 2023].

GeeksforGeeks. (2017). *Mutex lock for Linux Thread Synchronization - GeeksforGeeks*. [online] Available at: https://www.geeksforgeeks.org/mutex-lock-for-linux-thread-synchronization/.

GeeksforGeeks. (2022). *Implement thread-safe queue in C++*. [online] Available at: https://www.geeksforgeeks.org/implement-thread-safe-queue-in-c/.

Python Tutorial (N.D.). *How to Use Python Threading Lock to Prevent Race Conditions*. [online] Available at: https://www.pythontutorial.net/python-concurrency/python-threading-lock/.