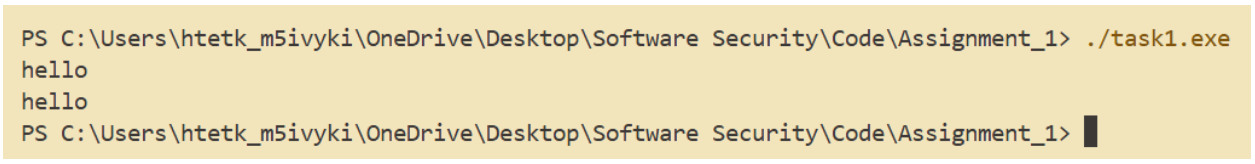
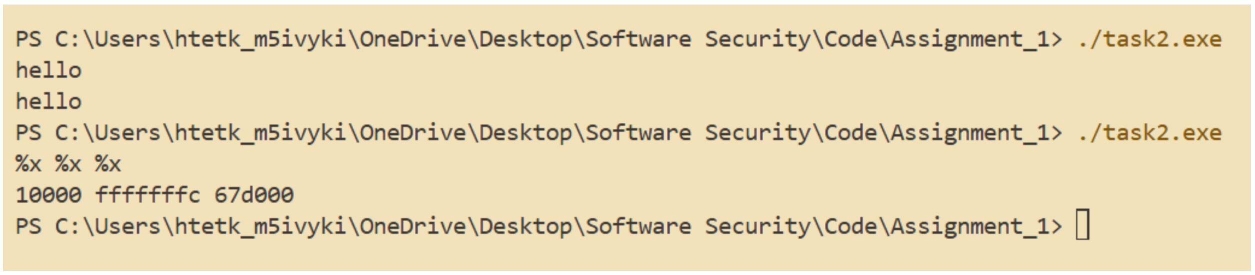
**Task 1**



```
PS C:\Users\htetk_m5ivyki\OneDrive\Desktop\Software Security\Code\Assignment_1> ./task1.exe
hello
hello
PS C:\Users\htetk_m5ivyki\OneDrive\Desktop\Software Security\Code\Assignment_1>
```

This program is safe because it uses fgets to read input which prevents buffer overflow.
In printf, the format specifier is used to avoid format string vulnerabilities.

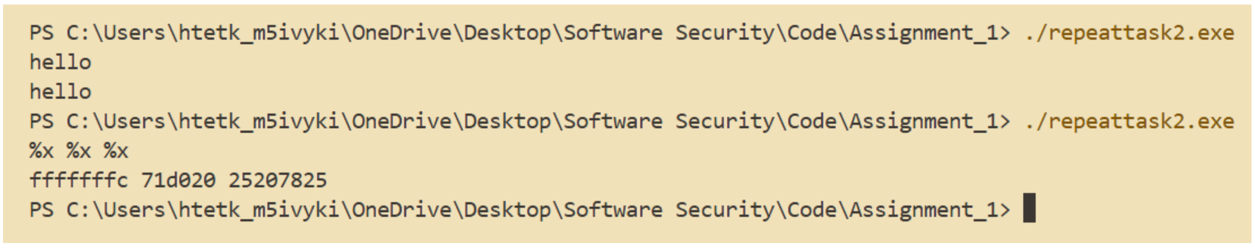
**Task 2**



```
PS C:\Users\htetk_m5ivyki\OneDrive\Desktop\Software Security\Code\Assignment_1> ./task2.exe
hello
hello
PS C:\Users\htetk_m5ivyki\OneDrive\Desktop\Software Security\Code\Assignment_1> ./task2.exe
%x %x %x
10000 fffffffc 67d000
PS C:\Users\htetk_m5ivyki\OneDrive\Desktop\Software Security\Code\Assignment_1>
```

The program outputs an unexpected behavior due to format string vulnerability.
It uses user input directly in printf without a format specifier.

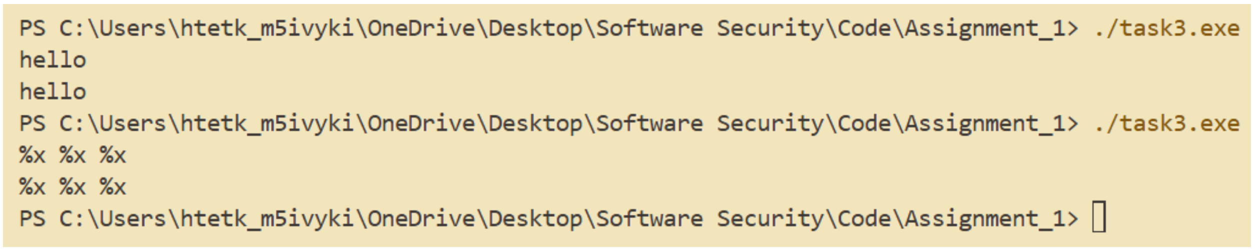
**Repeat Task 2**



```
PS C:\Users\htetk_m5ivyki\OneDrive\Desktop\Software Security\Code\Assignment_1> ./repeattask2.exe
hello
hello
PS C:\Users\htetk_m5ivyki\OneDrive\Desktop\Software Security\Code\Assignment_1> ./repeattask2.exe
%x %x %x
fffffffc 71d020 25207825
PS C:\Users\htetk_m5ivyki\OneDrive\Desktop\Software Security\Code\Assignment_1>
```

The program outputs an unexpected behavior due to format string vulnerability.
It uses user input directly in fprintf without a format specifier.
As it was with printf in task2.c, this unexpected behavior still exists.

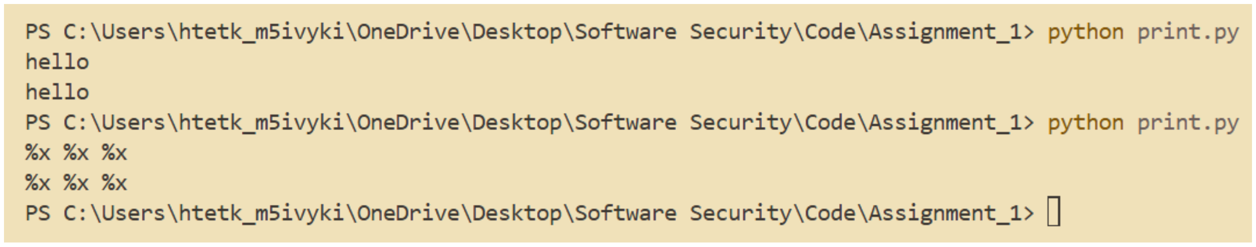
**Task 3**



```
PS C:\Users\htetk_m5ivyki\OneDrive\Desktop\Software Security\Code\Assignment_1> ./task3.exe
hello
hello
PS C:\Users\htetk_m5ivyki\OneDrive\Desktop\Software Security\Code\Assignment_1> ./task3.exe
%x %x %x
%x %x %x
PS C:\Users\htetk_m5ivyki\OneDrive\Desktop\Software Security\Code\Assignment_1>
```

The vulnerability in this program has been fixed by, by adding "%s" in printf, which is using a format specifier in printf.
This prevents format string vulnerabilities by ensuring that user input is treated as data rather than a format string.

**Task 4**



```
PS C:\Users\htetk_m5ivyki\OneDrive\Desktop\Software Security\Code\Assignment_1> python print.py
hello
hello
PS C:\Users\htetk_m5ivyki\OneDrive\Desktop\Software Security\Code\Assignment_1> python print.py
%x %x %x
%x %x %x
PS C:\Users\htetk_m5ivyki\OneDrive\Desktop\Software Security\Code\Assignment_1> 
```

User input here does not cause string vulnerability because python treat strings as data by default, rather than format strings.
Python separates data from executable code, so user input is not interpreted as executable code.