**Challenge Title: Deep Bug Gert**

**Challenge Description**:

If you can’t find the treasure, contact our trusted friend Deep Bug Gert. Follow his step-throughs till the end...

**Solution**:

This reverse engineering challenge focuses on anti-debugging techniques. Participants are provided with a Windows executable that checks whether it is being executed in a debugger environment. The flag is revealed only when the executable is run in a debugger. The challenge guides participants to explore the executable in Ghidra, understand the anti-debugging logic in the \_main function, and then use a debugger (WinDBG) to reveal the flag.

Theory about anti-debugging for better understanding:

In the context of malware and other malicious executables, the IsDebuggerPresent anti-debugging technique is employed to hinder analysis and reverse engineering efforts. Malicious code can incorporate this function to dynamically check if a debugger is attached to the process. If a debugger is detected, the malware may alter its execution flow, generate misleading outputs, or even terminate, making it more challenging for security analysts to analyze and understand its behavior. This helps malware evade detection and analysis in a controlled debugging environment, adding an extra layer of complexity for researchers.

**FLAG : FLAG{IsDebuggerPresent\_is\_an\_antiDebug\_technique}**

**Steps to find flag:**

Download and Install WinDBG:

Instruct participants to download and install a free debugger like WinDBG. Provide them with the necessary resources or direct them to the official WinDBG download page.

Open Ghidra:

Launch Ghidra and create a new project. Load the provided Windows executable into the project.

Navigate to \_main Function:

Locate and navigate to the \_main function within the Ghidra CodeBrowser window. Analyze the logic to understand the anti-debugging checks.

Identify Anti-Debugging Logic:

Identify the portions of code in the \_main function that check for the presence of a debugger. This might involve checking specific flags, registry values, or other indicators.

Load Executable in WinDBG:

Open the executable in WinDBG and set breakpoints or configure the debugger as needed.

Continue Execution:

Click on the "Run" button in WinDBG to continue the execution of the executable. Observe the debugger's output in the command prompt.

Flag Printing:

If the executable detects that it is being executed in a debugger, it will print the flag in the command prompt within WinDBG.