TryHackMe Challenge Writeup: Compiled

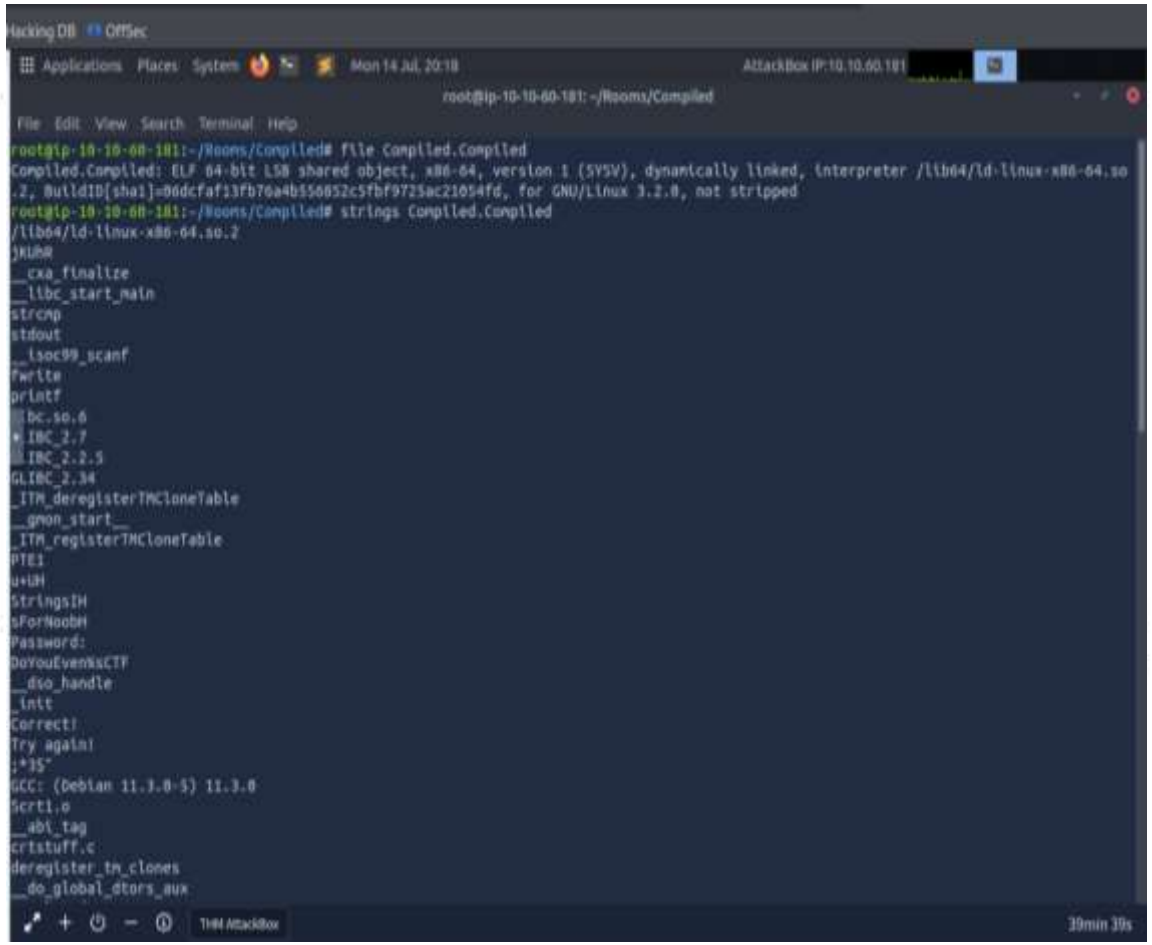
Introduction to the Compiled Challenge

Overview of the challenge and tools used

- **Challenge Overview:** The Compiled challenge presents a unique task, requiring the application of reverse engineering skills to uncover a hidden password embedded within a binary executable file.

- **Challenge Objective:** The primary goal is to locate and extract a concealed password from the compiled binary, demonstrating practical skills in analysis and problem-solving in cybersecurity.

- **Tools Utilized:** For this challenge, we leverage various analytical tools including 'strings' for string extraction, 'file' for basic file analysis, and 'Ghidra' for binary decompilation.



In this segment, we lay the foundational understanding of the Compiled challenge. Your task is to delve into a binary executable file, unearthing a password that is not readily visible. This challenge amalgamates various skills in reverse engineering and helps practitioners test their mettle against real-world cybersecurity scenarios. The tools we deploy in this endeavor—specifically 'strings', 'file', and 'Ghidra'—are core components of our toolkit, instrumental in unraveling the complexities of binary data.

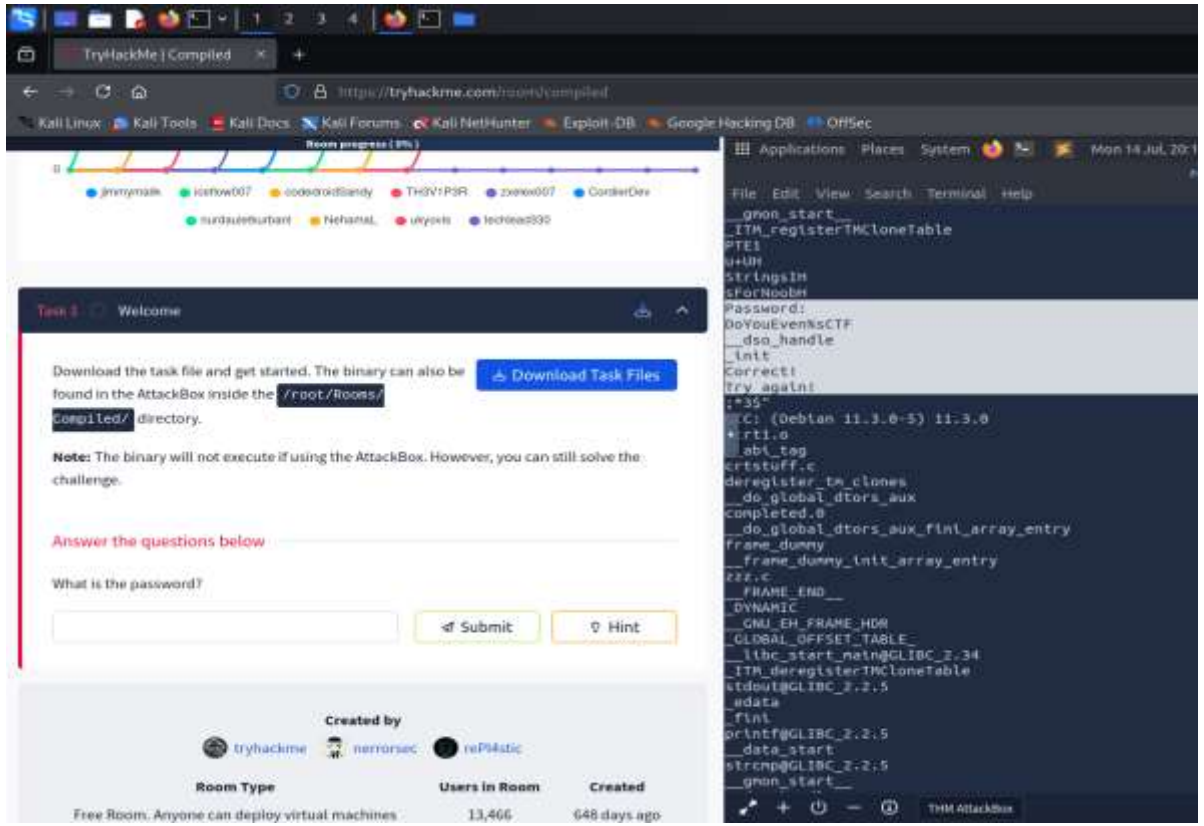
File Inspection

Understanding the binary structure

- **Initial Binary Analysis:** The binary file we start with is titled 'Compiled.Compiled', serving as our focal point for investigation as we seek to extract embedded secrets.

- **Using the File Command:** The 'file' command plays a critical role, providing essential details about the binary format, which in this case is identified as an ELF 64-bit shared object.

- **Output Analysis:** Interpreting the output informs our subsequent actions, elucidating the binary's architecture and potential vulnerabilities visible through its operational context.



Moving onto file inspection, we begin with our initial binary, 'Compiled.Compiled'. Utilizing the 'file' command enables us to dissect its structure and understand its underlying format. In this instance, we turn our attention to its identification as an ELF 64-bit shared object. This classification not only directs us toward how the binary might behave but also hints at the intricacies we need to navigate during our decompilation efforts.

Character Analysis with Strings Command

Extracting and identifying potential strings

Extracting Printable Strings

Using the 'strings' command, we can extract all human-readable strings from the binary. This essential step helps to surface pieces of textual data that may include our target password.

Identifying Potential Passwords

By analyzing the output for strings that resemble passwords or command sequences, we can potentially pinpoint the hidden credential we seek.

Challenges with Misleading Strings

While extracting useful strings, it's crucial to assess their context carefully, as some may be misleading or irrelevant to our objective.

In the fourth slide, we focus on character analysis utilizing the 'strings' command. This command is vital for extracting human-readable content from the binary, assisting us in our quest for any embedded strings that may act as passwords. However, we must remain vigilant against the allure of misleading strings, filtering through the output with a critical eye to uncover genuinely useful information.

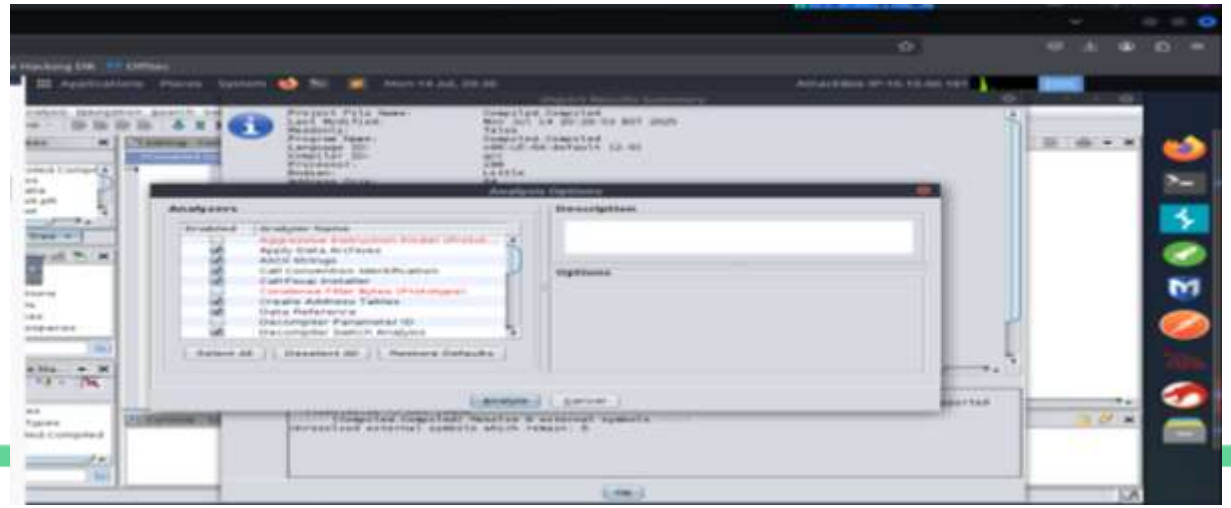
Decompiling with Ghidra

Setting up and analyzing the binary

Setting Up Ghidra Project
Establishing a Ghidra project is our next step, where we create a workspace that facilitates the analysis of the provided binary file.

Importing the Binary
The binary file 'Compiled.Compiled' is imported into Ghidra, enabling the tool to parse and prepare it for further decompilation and analysis.

Default Analysis Settings
We configure default analysis parameters, which streamline the examination of the binary, aiding in recognizing functions, variables, and control flows within the code.



As we transition into the realm of Ghidra, we commence with the setup of our analysis project. The initial step involves importing our binary file into Ghidra, which allows the tool to dissect the binary structure. By setting default analysis parameters, we prepare ourselves to identify critical functions and variables seamlessly, equipping ourselves for the deeper exploration that lies ahead.

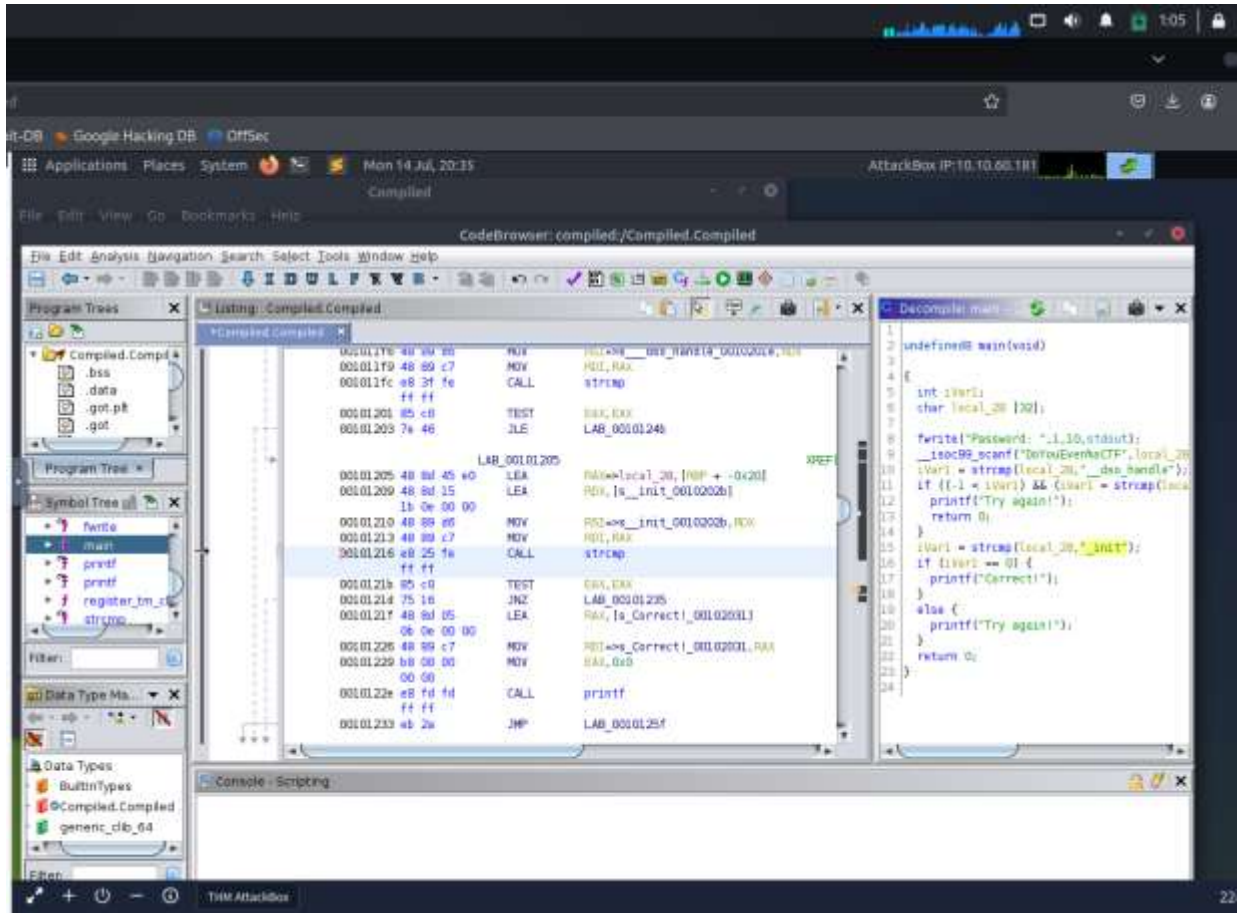
Searching for Key Functions

Locating essential function calls

- **Importance of strcmp:** The 'strcmp' function emerges as a pivotal component in validating user input against the correct password during the authentication process.

- **Locating strcmp in Decompiled Code:** We utilize Ghidra's search functionalities to pinpoint instances of 'strcmp' within the decompiled code, crucial for understanding flow and verification mechanisms.

- **Understanding Function Usage:** Analyzing how 'strcmp' is employed provides insight into the logic behind password matching, leading us closer to our objective.



In the sixth slide, we delve into the function analysis, honing in on 'strcmp', which verifies user input against the predefined credential. This function's significance lies in its role in the security framework, ultimately determining access rights to the system. By locating this function within the decompiled code, we gain critical insights into how user input is processed, anywhere from comparisons to control flow.

Analysis of Decompiled Code

Deciphering the logic behind validation

Identifying the Hardcoded Password

Within the decompiled analysis, we successfully identify the hardcoded password, cementing our progress in the challenge.

Comparing User Input

The logic of comparing user-entered data to the identified password underscores the simplistic yet effective nature of binary authentication.

Significance of Direct String Comparison

Examining how direct comparisons are made offers clarity into the binary's design, emphasizing the need for precision in authentication mechanisms.

Focusing our efforts on the analysis of the decompiled code allows us to trace the logic behind the password validation process. Our identification of the hardcoded password marks a significant milestone in the challenge. Understanding how user input is compared against this embedded string also opens our eyes to potential security weaknesses tied to such direct comparisons, urging the importance of secure programming practices.

Final Result and Execution

Executing with the discovered password

Running the Binary

Upon entering the revealed password, we execute the binary, which triggers the intended sequence of events, finalizing our challenge.

Password Discovered

The hidden credential we have extracted is 'DoYouEven_init', a humorous nod to popular culture that illustrates their choice in passwords.

In concluding the challenge, we highlight the critical final step: executing the binary with the password we unveiled. The revelation of the password 'DoYouEven_init' not only serves the task at hand but also adds a layer of amusement reflective of contemporary culture. The successful execution affirms our understanding of reverse engineering principles, tying together the various stages of analysis and exploration into a coherent outcome.