Program Description

The attack leverages a stack buffer overflow vulnerability to manipulate the execution flow of a program:

Identifying the Vulnerability:

- The fgets library routine reads data from the network connection using an incorrect length parameter, leading to a stack buffer overflow.

Shell Code Preparation and Call:

- Shell code is designed with a limited size due to the constraints of the vulnerable buffers.
- The objective is to execute the */bin/env* command on the target server and send the environment variables back to the attacker.
- The *execve* system call is chosen for its ability to replace the current process with a new one, allowing the execution of arbitrary commands.
- The x64 architecture's registers (e.g., *rax, rdi, rsi, rdx*) are utilized to pass parameters for the system call.
- The length of the string used to compromise the server is 160 and the location of the return address on the stack is 0x3e, 0xe0, 0xff, 0xff, 0xff, 0x7F.

Assembly Language Code:

- The shell code is written in assembly language to ensure it is position-independent and free of newlines or null characters.
- The assembly code sets up the stack, loads the address of the environment variable, and prepares for the *execve* system call.
- Special attention is given to handling null bytes to avoid issues in the exploitation process.
- NOP (No-Operation) instructions (0x90) are used for padding to align the return address properly. This aligns with the *MNOPWXYZ* string used to crash the program.
- Padding with NOP instructions allows flexibility in the placement of the return address on the stack, which is particularly useful when address randomization is enabled.

Exploit Execution:

- The attacker sends a meticulously crafted payload to the target server, causing a stack buffer overflow.
- The return address is overwritten with the address of the shell code, redirecting the program's execution flow.

- The altered return address leads to the execution of the shell code, initiating the */bin/env* command.
- The environment variables are then sent back to the attacker, achieving the data theft objective.

In summary, the attack manipulates a stack buffer overflow to inject and execute custom shell code, exploiting a vulnerability in the server's input handling mechanism.