pwv binary vulnerability analysis

Introduction

I really enjoyed analyzing this binary, and found it quite challenging, at least compared to some of the other crackmes I've done. However, because of the uncertain outcome of the challenge and me looking for a job ASAP, I couldn't dedicate too much time on the analysis, and did not explore it in the same depths I probably would have if this was my actual job (I hope this is understandable). As such, I did not come up with any concrete results as I would've hoped. That's why I decided to also include my thought process and some of the false positives I encountered, especially so considering this is to determine a candidate's abilities and not for a client.

Analysis

After reverse engineering the binary with ghidra and determining some of its core technologies (pthreads, message queues, getspent etc.), I started looking for all possible inputs. I must say that once I saw the signal handler function, I hoped for a vulnerability through that (signal handlers often end the program abruptly before any possible root privileges are revoked), which wasn't the case.

It quickly became clear that, except for the numerical 'workers' command-line argument, the only other input was the `/etc/shadow` file, itself. The worst case scenario would be if a non-root malicious user was able to indirectly edit the file, perhaps through the creation of a new user with `useradd` and was able to overflow the buffer available for the name of the user. Such a vulnerability would allow for memory corruption or even arbitrary code execution. Unfortunately, the buffer provided for `name` in the binary is 33 bytes, which covers for the Unix max 32 character name and the null terminator.

Another possible way to edit `/etc/shadow` would be with `passwd`, since Unix has no limit on the password size and the binary provides 1031 bytes. The passwords, however, are hashed into a fixed-length format, which would have to be cracked in order for the password to ever be stored into those 1031 bytes. That means that the shellcode and all the buffer would have to actually be cracked by `pwv`, and not only that, but from the description of `pwv`, it doesn't even attempt to crack passwords longer than 40 bytes (32 bytes for the name + '1234' etc., and the 4-digit number passwords it tries are even less). Another issue with this approach is that, both `useradd` and `passwd` are commands accessible only to root, so they would have to be `setuid` for anyone else to exploit a `pwv` vulnerability through them, which would already be a serious system vulnerability.

The only other way for a user to edit `/etc/shadow` would be if they had direct access to it, which would, again, be a major flaw.

Conclusion

Unfortunately, due to my current limitations I did not have the opportunity to actually properly test the above. My final, very-rushed conclusion would be that `pwv`, given the fact that it's a program accessible to root only, can provide realistic attack vectors to malefactors only if there are other (perhaps more critical) vulnerabilities in the system in which it is used. I cannot imagine a scenario in which it would constitute a threat by itself.

I'd be intrigued if that actually wasn't the case, and it must go without saying that I'm interested in the source code of `pwv`, as well as the challenge answers. I would also really appreciate honest feedback on my thought process and how well (or how badly) I performed!

Michael Constantine Dimopoulos

Email: mk@mcdim.xyz