Web Post Client

Description

This lab aims to practice implementing a web client to upload a file to an internal web server. Please follow the instructions and have fun!

1. While we request you to upload a file to an internal web server, a public version of the server is available at http://inp.zoolab.org:10314/ for you to test.
   The public server behaves exactly the same as the internal one, except that it ***does not log a verified operation***. You must interact with the internal web server to ensure the internal server can successfully log your operation.
2. The server provides several APIs for you to interact with, as listed below. To play with the public server, assume the SERVER is at:
   http://inp.zoolab.org:10314.
   To interact with the real server, replace the server name and port number with what you retrieved from the $SERVER/addr API.
   - **$SERVER/addr**: Get the internal server's IP address and port number.
   - **$SERVER/logs**: Check if the server logs your operation successfully.
   - **$SERVER/otp?name=<your-student-id>**: Get an one-time-password (OTP) token from the server. Please replace <your-student-id> with your student ID.
   - **$SERVER/verify?otp=<OTP>**: Check whether your current OTP is valid. Note that the OTP passed to this URL must be a valid URL-encoded string (for some special symbols). **This interface is for debugging purposes only. You have to pass this lab by uploading your OTP via the $SERVER/upload interface introduced later.**
   - **$SERVER/upload**: The primary interface for validating your OTP. Ideally, you have to save your obtained OTP in a text file and then upload it to the server. **However, because you cannot access any storage spaces in our runtime sandbox (introduced later)**, it would be better to handle everything in memory without writing any files.
     This URL supports two HTTP methods, GET and POST. For the former one, it simply displays a web interface for a user to choose the file to upload and submit the file. For the latter one, it accepts the file uploaded by the user. You may use tcpdump, wireshark, and browser developer tools to inspect how the client and the server interact.
   - **$SERVER/logs**: A valid OTP file uploaded via the uploading interface will be recorded and can be displayed by accessing this interface.

   You can play with the public server first and implement your solution to interact with the server. Once you are satisfied with your implementation, you can then

upload your implementation to our sandbox to interact with the internal one.

3. You have to implement your solution in C or C++. The implementation should follow the steps.
    - The server's IP address and port number (either public or internal) can be hard-coded in your program.
    - Your program has to obtain an up-to-date OTP via the **$SERVER/otp?name=<your-student-id>** interface.
    - Your program then sends the OTP to the internal server via the **$SERVER/upload** interface.
    - If everything works well, you should receive an OK message from the server. You can then check the **$SERVER/logs** interface to see if the uploaded record is successfully logged in the server.
4. To access the internal server, you must upload your compiled binary executable to our ***sandbox server***. The sandbox server runs your program on a host that can connect directly to the internal server.
   The sandbox server can be accessed via
   nc inp.zoolab.org 10315
   Your executable must be compiled and linked with the -static option, and send the entire binary to the sandbox server as a BASE64-encoded zlib-compressed string, i.e., b64encode(zlib.compress(payload)). To simplify the upload process, we prepare a submission script implemented with the pwntools package. You can download the submission script from here (view).
5. Sample curl commands for interacting with the public server. Assume the username is aaa.
   curl --silent 'inp.zoolab.org:10314/otp?name=aaa' >! otp1.txt
   curl -F 'file=@otp1.txt' 'inp.zoolab.org:10314/upload'
6. Our runtime platform currently supports both x86_64 and aarch64 binary file. You can send statically linked executables compiled in the debian docker running on x86_64 and aarch64 CPU.
   If you still want to perform cross-compiling on aarch64-based machine, you can setup a cross-compilation environment by following the instructions below.
   dpkg --add-architecture arm64
   apt-get update
   apt-get install gcc-x86-64-linux-gnu g++-x86-64-linux-gnu
   The compilation command is
   x86_64-linux-gnu-gcc <source-code.c> -o <output> -static
   If you implement your solution in c++, you may need to create symbolic links for the missing libraries using the command:

cd /usr/lib/x86_64-linux-gnu/; ln -s /usr/x86_64-linux-gnu/lib/* .

Demonstration

1. [10%] You have pwntools installed successfully. Installation instructions can be found here. To verify your installation, run the following codes in your python3 runtime.

   from pwn import *

   import random

   z = random.randbytes(16)

   context.arch, context.bits, = 'amd64', 64, print(disasm(z))

   context.arch, context.bits, = 'i386', 32, print(disasm(z))

2. [10%] Implement a program that simply prints hello, world! and upload it to run on the sandbox server.
3. [50%] Your implementation works with the public server. Your implemenation must connect to the server using socket APIs and cannot use system or similar functions in your implementation.
4. [30%] Your implemention can be uploaded to the server and interact successfully with the server. Once it's done, check the $SERVER/logs to see it has successfully passed the server verification.