EJERCICIO OBLIGATORIO -PRÁCTICA 4

Prueba de carga utilizando Jmeter y vamos a atacar una API Rest.

1. Instalar DockerEngine sobre una máquina virtual (ubuntu)
2. Desde la misma máquina virtual donde descarguemos eso, nos clonamos el repositorio del ejercicio (**iseP4jmETER**)
3. ls (para comprobar que se ha clonado el repositorio)
4. docuker compose up (la primera vez que lo descaguemos tardará mucho porque tiene que bajarse muchas cosas).

> Una característica de los contenedores es que una vez que descargas, en futuras ejecuciones no hace falta volver a descargar todo.

5. Se hace ssh a la misma máquina desde otra terminal (2 terminales de la misma máquina aunque se podría hacer con la máquina del host anfitrión)
6. Hacemos ./pruebaEntorno.sh y veremos en la otra pantalla cosas.
7. Abrimos un browser y escribimos: http://192.168.56.15:3000

   Nos tendrá que salir **ETSIIT Alumnos API .....**
8. Veremos un login y una contraseña.. ¿dónde cogemos esas credenciales: nos metemos en el repositorio --> carpeta JMeter --> administradores.csv Una vez loggeados nos dará un Token como resultado, que se tendrá que indicar a la hora de hacer la petición **GET ...**
9. Con el protocolo de seguridad BasicAuthEl cliente tiene que saber unas credenciales
10. Abrimos el archivo **pruebaEntorno.sh**:

    SERVER="servidor que vamos a atacar"

    TOKEN= hacemos llamada a curl (para hacer llamadas http)

    ```
    -u
    -X POST://$SERVER:3000/api/vi/auth/login --> hacemos una petición al
    endpoint indicado
    ```

> Una petición sin verbo se enntiende como **GET**

11. Si ejecutamos otra vez, y copiamos todo el chorro de cosas que nos da (nos da la información en formato **JSON**); si queremos ver mejor el contenido, vamos a un printer online de JASON.
12. Si vamos a la página web "JSON Web Tokens" podemos ver información interesante sobre estas cosas, lista de claims, etc.

> Los tokens no se cifran aunque sí está firmados porque lo que nos interesa qué información no ha sido manipulada con nadie

13. En el apartado Debugger de esa página podemos hacer cosas para verificar firmas, etc.

14. Una aplicación de contenedores en la mayoría de los casos está compuesta por varios contenedores. En nuestro caso, tenemos un contenedor volátil (MongoInit) que lo que hace es que cuando arranca, envía los datos y se muere (los envía al contenedor Mongo (mongodb)). Lo normal es que los conetendedores no tengan almacenamiento permanente entonces lo que hacemos es tener los datos en otro sitio. Lo que hacemos con mongodb es poner un puerto, y Node (otro contenedor) lo consulta. Node expone a travéss de su puerto 3000 /login y /passwd.

15. Un DOCKER tiene el siguiente formato:

    mongodb: image: mongo (image = repositorio donde la gente publica sus contenedores) ports: - "27017:27017"

    o

    nodejs: build: ./nodejs (directorio nodjs del repositorio dentro del cual hay un fichero (**DOCKERFILE**)que contiene el código fuente para la construcción de la imagen de docker)

    DOCKERFILE

    FROM = desde donde se va a contruir el docker RUN = cuando ya hayas hecho eso, crea un directotio COPY= y copia el cotenido en el directorio creado ... CMD ["npm", "start"] arranca el servicio

16. En la página web "hub.docker.com" nos podemos bajar dockers

17. Vamos a empezar a trabajar con JMeter

    18. Arrancamos JMeter (que necesita java) y vamos a atacar los dos endpoints (login y expediente)

    19. Test Plan = base de todo proyecto JMeter. Completar datos de Test Plan

> Para cambiar el idioma Options -> Language

    20. Boton derecho (del archivo que se ha creado al aceptar lo del Test Plan) **Thread Group**. El thread group se utiliza para gestionar grupos de usuarios:

        Name: Alumnos

        - Continue

        Thread Properties

        ```
        Number of threads (users): 10
        Ramp-up period (seconds): 1  (porque vamos a ejecutar pocas
        hebras)
        Loop Count: 5  (número de veces que se van a ejecutar dichas
        hebras)
        Same user on each iteration
        ```

    21. Vamos a hacer una prueba de http (botón derecho: add -> sampler -> HTTP REQUEST):

Nombre: Pagina Home de UJA

Protocol | Sever name or IP: www.ujaen.es (la ip siempre **sin** el protocolo (http o htpps)) Si la dirección tiene cosas con /.../... esto lo tenemos que poner en el apartado "path"

HTTP Request:

Boton derecho add listener (añadimos los listener "View Results Tree" (no tener activo a la hora de correr", y "Aggregate Report"))

Para ver los resultado: doble click en "View Results Tree" en el apartado de "Text"

22. Hacemos una HTTP Request y la llamamos Login Alumnos.

    IP: 192.168.56.15 Port: 3000 HTTP Request: POST Path: /api/v1/auth/login

    Parameters:

    ```
    name (login) value (el correo que se coge del fichro alumnos.csv)
    el basicauth
    ```

    Boton derecho add --> config nose que --> HTTP Authorization Manager. Le ponemos de nombre "Basic Auth ETSII:

    ```
    Base URL (htpp://192.168.56.15) -> username --> password -->
    BASIC

    Vemos los usuarios
    ```

23. HTTP Request para el expediente

    Nombre: Consultar Expediente de Alumno

    IP: 192.168.56.15 Port: 3000 HTTP Request: GET PATH: /api/v1/alumnos/alumno

    Guardamos y le damos a ejecutar. Nos sale error en autenticación porque no le hemos pasado la cabecera del token (ese -H "Auth..." del fichero pruebaEntorno). Psra ello Boton dercecho --> add -> configuration... -> Header Manager. Pegar la cabecera del token en el campo "Value"

> En el TEST PLAN podemos crear variiable con objeto de no esta escribiendo todo el rato en las pruebas que hagamos y evitar así equivocarnos. Simplemente le damos a **add** y para indicar que use la variable en un campo dado, hacemos simplemente **${VARIABLE}**. Hacer esto para la IP y el PUERTO por ejemplo. Podemos hacer un configure "Valores por defecto" en el que indicamos que todo lo que haya debajo de dicho archivo tenga el mismo valor de las variables que se le haya indicado para evitar así tener que estar escribiendo los valores de las variable en todas las pruebas que hagamos.

24. Para lo administradores lo mismo pero:

add sampler Access log

25. Los timers se encuentran en: add -> Timer -> ...

> TENEMOS QUE LANZAR LA PRUEBA CON EL COMANDO:

```
jmeter -n -t [jmx file] -l [results file] -e -o [Path to web report
folder]

jmeter -n -t p4_clase.jmx -l resultados.jlt
```

> LOS PATH DE LOS ARCHIVOS SE PONEN RELATIVOS: en el campo "filename", cuando le damos a "browse" y le damos a abrir un archivo, el filename se r

> SI QUEREMOS EJECUTAR LA PRUEBA CON EL ENTORNO GRÁFICO, EN LUGAR DE ABRIRLA CON EL ICONO DE LA "carpeta arriba en el menu", PODEMOS DARLE A:

```
HA CAMBIADO DE VENTANA Y NO ME HE ENTERADO !!!!!!!!!!!!!!
```

Consideraciones finales

Usar variables Result tree desactivado Paths relativos. Enviarle el fichero jmk y el fichero con resultados

Apache Benchmark

Se instala por defecto con el httpd aunque se puede instalar a parte. Se utiliza para hacer pruebas de carga sobre servidores http.

Utilización: 10 llamadas con una concurrencia de 2 hebras :

> ab -n 10 -c 2 http://www.ugr.es/

En el protocolo http cuando algo nos da una respuesta correcta, el código de respuesta es 2xx. Vemos lo que no devuelve el comando anterior, en el apartado:

Non-2xx respones: 10 (le mosquea porque todas las 10 llamadas no son exitosas)

Hacemos una llamada con curl:

curl -v http://www.ugr.es

Veremos que en location, aparece http**s** (se redirige hacia https)

Lo estamos haciendo mal entonces. Tenemos que poner:

> ab -n 10 -c 2 https://www.ugr.es/

Ahora ya sale bien.

```
    Estadísticas Connection time:

    Connect: Tiempo que tardamos en llegar al servidor
    Processing: Tiempo que transcurre entre desde que le mandamos la
petición get hasta que recibimos la respuesta.
    Waiting
    Total: suma de Connect y Processing (se obvia el waiting)

    Percentiles:

        Cuando decimos que un servidor es capaz de procesar 20000 por
minutos, el tiempo de latencia es de 145ms, etc. cuando hacemos
afirmaciones así, estamos diciendo que la latencia,... está en el percentil
x%
```

> DE ESTO PUEDE PREGUNTAR MÉTRICAS (CONOCER TODAS LAS MÉTRICAS DE APACHE BENCHMARK) Y PERCENTILES.

Phronix

Aplicación para correr benchmark similar a una página de github donde la gente sube proyectos opensource. Para insalarlo, la forma más cómoda es mediante la imagen de Docker: phronix/pts Docker image | Docker hub (buscarlo en un browser)

(los contenedores no virtualizan por eso puedo correr phroniix sobre docker.)

1. docker run -it phronix/pts (correr docker con interfaz interactiva)
2. phronix-test-suite <opcion> (los suite son agrupaciones varios benchmark disponibles)
3. Para ver los suites dispinibles:

   phronix-test-suite list-available-suites
4. Al profesor le daba por culo docker ps muestra los contenedores de docker que están ejecutandose. Nos podemos conectar a un contenedor con:

   docker exec -it ID_CONTENEDOR /bin/sh

   cd phronix-test-suite

   ./phronix-test-suite list-avaiable-suites

   ./phronix-test-suite install pts/compress-gzip

   ./phronix-test-suite benchmark pts/compress-gzip