



TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE TLAXIACO

Seguridad y Virtualización

Practica 1

Integrantes del Equipo:

Rael Gabriel Bautista

Sandra Gabriela Velasco Guzmán

Amilkar Vladimir Reyes Reyes

Arnol Jesus Cruz Ortiz

Docente:

Edward Osorio Salinas

Carrera:

Ingeniera en Sistemas Computacionales

Grupo: 7US

Semestre: Agosto – diciembre 2024

30/Agosto /2024

1.- Crea un programa en algún lenguaje de programación que permita al usuario ingresar una contraseña y que valide si la contraseña es segura o no. Una contraseña segura debe cumplir con los siguientes criterios basados en las recomendaciones de Google:

PROGRAMA 1

A continuación, se muestra nuestro primer código funcional en Python.

Nuestro código lo que hace es que permita al usuario ingresar una contraseña y que valide si la contraseña es segura o no mediante ciertos criterios de validación.

Entonces lo primero que se tuvo que realizar es la definición de la función y se le puso contraseña segura y esta servirá para poder verificar si la contraseña es segura o no.

```
3 def es_contraseña_segura(contraseña):
```

Entonces lo que hace esa función es poder validar los siguientes criterios.

- Tener al menos 8 caracteres.
- Tener al menos una letra mayúscula (A-Z).
- Tener al menos una letra minúscula (a-z).
- Tener al menos un número (0-9).
- Tener al menos un carácter especial (!, @, #, \$, %, ^, &, *, (,), -, _, =, +, [,], {, }, |, \, ;, :, ', ", ,, ., <, >, /, ?, ~, `).
- No debe contener espacios en blanco.

Para cada criterio, se utiliza la función `re.search()`, que busca el patrón especificado en la contraseña y devuelve `True` si encuentra una coincidencia.

```
3 def es_contraseña_segura(contraseña):
4     # Criterios de validación
5     longitud = len(contraseña) >= 8
6     mayuscula = re.search(r'[A-Z]', contraseña)
7     minuscula = re.search(r'[a-z]', contraseña)
8     numero = re.search(r'[0-9]', contraseña)
9     caracter_especial = re.search(r'[@#$%^&+=]', contraseña)
```

Ya después tuvimos que poner otro criterio donde no debe tener más de 2 caracteres iguales consecutivos.

La expresión regular que se utiliza es `r'(\1)'` funciona de la siguiente manera:

- `.` 'coincide con cualquier carácter.
- `\1` se refiere al mismo carácter encontrado por `.`

```
12
13 sin_repetidos = not re.search(r'(\1)', contraseña)
```

Para la validación final usamos una función donde combina todos los criterios en una única condición. Si todos los criterios son True, la función devuelve True, indicando que la contraseña es segura. Si alguno de los criterios falla, devuelve False, indicando que la contraseña no es segura.

```
16
17 if longitud and mayuscula and minuscula and numero and caracter_especial and sin_repetidos:
18     return True
19 else:
20     return False
```

Y ahora lo que utilizamos fue un input para que el programa solicite al usuario que ingrese una contraseña.

```
24 contraseña = input("Ingresa una contraseña: ")
```

El código final valida la contraseña usando la función `es_contraseña_segura()` y muestra un mensaje al usuario indicando si la contraseña es segura o no y así es como termina nuestro código.

```
27
28 if es_contraseña_segura(contraseña):
29     print("La contraseña es segura.")
30 else:
31     print("La contraseña no es segura. Asegúrate de que tenga al menos 8 caracteres, incluya mayúsculas, minúsculas, número")
32
```

Resultados

Aquí muestra que la contraseña es segura

```
Escritorio/seguridad y Virtualizacion/sguro.py"
Ingresa una contraseña: Amilkar@1
La contraseña es segura.
PS C:\Users\reyes\OneDrive\Escritorio\seguridad y Virtualizacion> █
```

Y si no cumple con los siguientes requisitos nos mandara un mensaje de que no es segura.

- Tener al menos 8 caracteres.
- Tener al menos una letra mayúscula (A-Z).
- Tener al menos una letra minúscula (a-z).
- Tener al menos un número (0-9).
- Tener al menos un carácter especial (!, @, #, \$, %, ^, &, *, (,), -, _, =, +, [,], {, }, |, \, ;, :, ', ", ,, ., <, >, /, ?, ~, `).
- No debe contener espacios en blanco.
- No debe tener más de 2 caracteres iguales consecutivos.

```
Ingresa una contraseña: ad,jjgf12
La contraseña no es segura. Asegúrate de que tenga al menos 8 caracteres, incluya mayúsculas, minúsculas, números y caracteres especiales.
PS C:\Users\reyes\OneDrive\Escritorio\seguridad y Virtualizacion> █
```

2.- Crea un programa que me recomiende una contraseña segura. La contraseña debe cumplir con los criterios de la instrucción anterior.

PROGRAMA 2

Ahora debemos de realizar un programa que nos recomiende una contraseña segura. La contraseña debe cumplir con los criterios de la instrucción anterior.

En este caso utilizamos un `get_random_char(charset)` que selecciona y devuelve un carácter aleatorio del conjunto `charset` pasado como argumento.

```
1 import random
2 import string
3 import tkinter as tk
4 from tkinter import messagebox
5
6 def get_random_char(charset):
7     return random.choice(charset)
```

También se ocupo un `shuffle_string(s)` la que se encarga de mezcla aleatoriamente los caracteres de la cadena `s` para eliminar cualquier patrón predecible.

```
def shuffle_string(s):
    s_list = list(s)
    random.shuffle(s_list)
    return ''.join(s_list)
```

Despues realizamos lo que viene siendo el generador de contraseñas en este caso se utilizo `generate_password()` es la que genera una contraseña, `length = 12`:esta sirve para colocar la longitud de caracteres para la contraseña ahí dependiendo a las necesidades la podemos modificar, `charset`: es el conjunto de caracteres posibles para la contraseña (letras mayúsculas y minúsculas, dígitos y símbolos especiales). Incluye al menos un carácter de cada tipo: Se asegura de que la contraseña tenga al menos una letra minúscula, una letra mayúscula, un dígito y un símbolo especial. También en esta parte de código se rellena el resto de la contraseña: Los caracteres restantes se completan de forma aleatoria después mezcla los caracteres: Para evitar cualquier patrón en la contraseña, se mezcla el orden de los caracteres, y al ultimo se utiliza `result_label.config(...)` que es la que muestra la contraseña generada en la interfaz gráfica.

```

14 def generate_password():
15     length = 12
16     charset = string.ascii_letters + string.digits + "!@#$%^&*(),.?\\\":{}|<>"
17     password = ""
18
19     # Asegura que la contraseña incluya al menos un carácter de cada tipo
20     password += get_random_char(string.ascii_lowercase)
21     password += get_random_char(string.ascii_uppercase)
22     password += get_random_char(string.digits)
23     password += get_random_char("!@#$%^&*(),.?\\\":{}|<>")
24
25     # Llena el resto de la contraseña con caracteres aleatorios
26     for _ in range(4, length):
27         password += get_random_char(charset)
28
29     # Mezcla los caracteres para evitar patrones predecibles
30     password = shuffle_string(password)
31
32     # Mostrar la contraseña generada
33     result_label.config(text="Contraseña segura recomendada: " + password)

```

Y ya por último hicimos una interfaz gráfica donde se abre una ventana y se vea mejor ya que también lo podemos abrir desde la terminal pero decidimos hacerlo de la siguiente manera, primero inicializamos la ventana principal, se establece un título a la ventana después se crea un contenedor ahí ponemos el color que deseamos de fondo, después creamos una etiqueta que muestre el título, procedemos a crear un botón que se utilizara para crear la contraseña, de ahí se crea una etiqueta donde se mostrara la contraseña creada y para que todo esto funcione se crea un bucle principal de la aplicación.

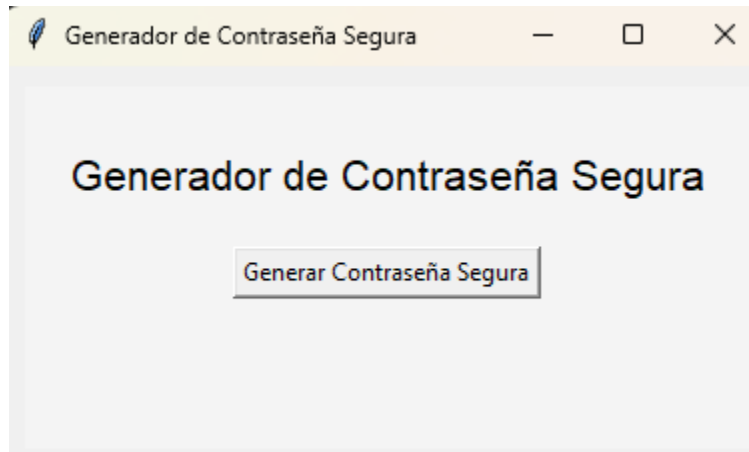
```

36 root = tk.Tk()
37 root.title("Generador de Contraseña Segura")
38
39 container = tk.Frame(root, padx=20, pady=20, bg="#f4f4f4")
40 container.pack(padx=10, pady=10)
41
42 title_label = tk.Label(container, text="Generador de Contraseña Segura", font=("Arial", 16), bg="#f4f4f4")
43 title_label.pack(pady=10)
44
45 generate_button = tk.Button(container, text="Generar Contraseña Segura", command=generate_password)
46 generate_button.pack(pady=10)
47
48 result_label = tk.Label(container, text="", font=("Arial", 12, "bold"), fg="green", bg="#f4f4f4", wraplength=300)
49 result_label.pack(pady=10)
50
51 root.mainloop()
52

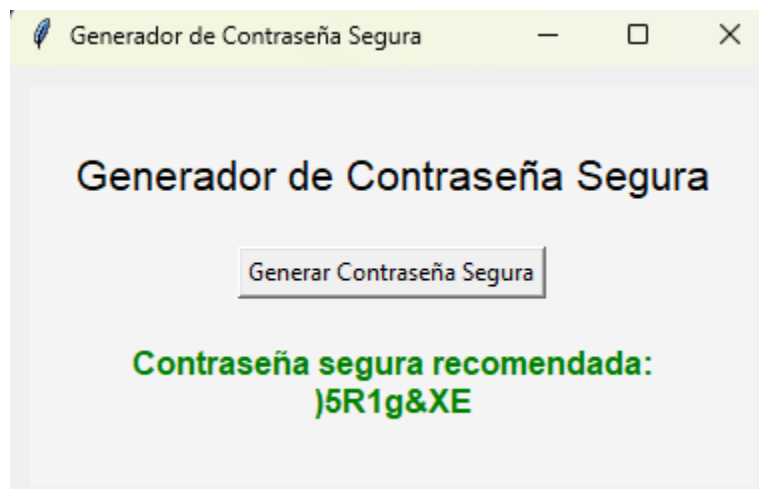
```

Resultados

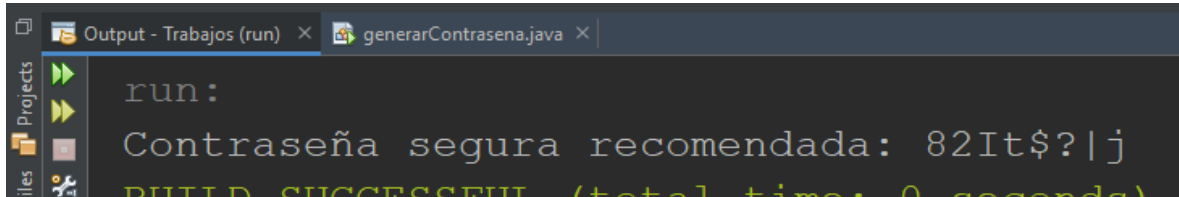
Para poderlo ejecutar nos dirigimos a donde dice ejecutar Python y de ahí ejecutar el archivo en Python y nos mostrara el programa que se realizo.



Ya que nos apareció le damos en generar contraseña segura y nos mostrara la contraseña que genera automáticamente.



3.- Crea un certificado SSH, clave pública y clave privada, añade el certificado SSH a tu cuenta de GitHub y realiza un git clone de un repositorio nuevo



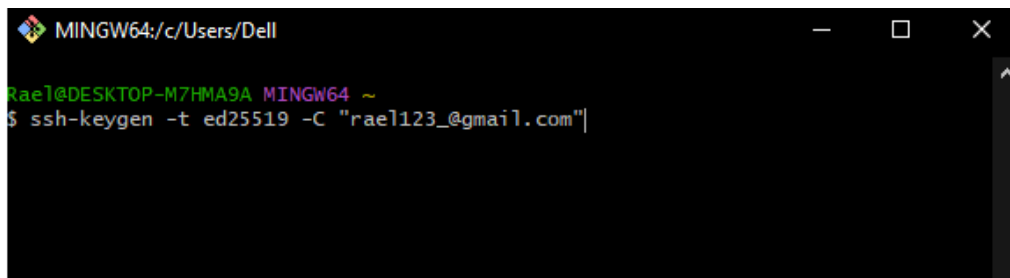
```
run:
Contraseña segura recomendada: 82It$?|j
BUILD SUCCESSFUL (total time: 0 seconds)
```

utilizando la ruta SSH del repositorio.

Un certificado SSH, que en este contexto se refiere a un par de claves SSH (una clave pública y una clave privada), sirve para autenticarte de manera segura con servicios remotos, como GitHub, sin necesidad de ingresar tu contraseña cada vez que interactúas con ellos.

1. Generar un par de claves SSH

- Para generar la clave utilizamos git bash ingresando el siguiente comando:

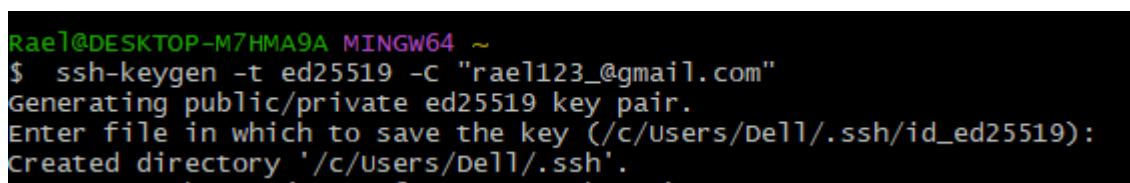


```
MINGW64:/c/Users/Dell
rael@DESKTOP-M7HMA9A MINGW64 ~
$ ssh-keygen -t ed25519 -C "rael123_@gmail.com"
```

- El parámetro -t ed25519 especifica el tipo de clave a generar (es preferible usar ed25519 por seguridad, pero también se puede usar rsa).
- Se debe ingresar el email asociado a nuestra cuenta de GitHub.

2. Guardar las claves

- Elegimos un nombre para el archivo donde se guardará la clave privada. Se puede presionar Enter para aceptar el nombre por defecto (id_ed25519).



```
rael@DESKTOP-M7HMA9A MINGW64 ~
$ ssh-keygen -t ed25519 -C "rael123_@gmail.com"
Generating public/private ed25519 key pair.
Enter file in which to save the key (/c/users/Dell/.ssh/id_ed25519):
Created directory '/c/users/Dell/.ssh'.
```


- Luego, ingresamos una frase de contraseña para la clave (opcional). Si se prefiere no usar una, simplemente presionar Enter

3. Copiar la clave pública al portapapeles:

- Ejecutamos el siguiente comando para copiar la clave pública al portapapeles:

```
Rael@DESKTOP-M7HMA9A MINGW64 ~  
$ cat ~/.ssh/id_ed25519.pub | clip
```

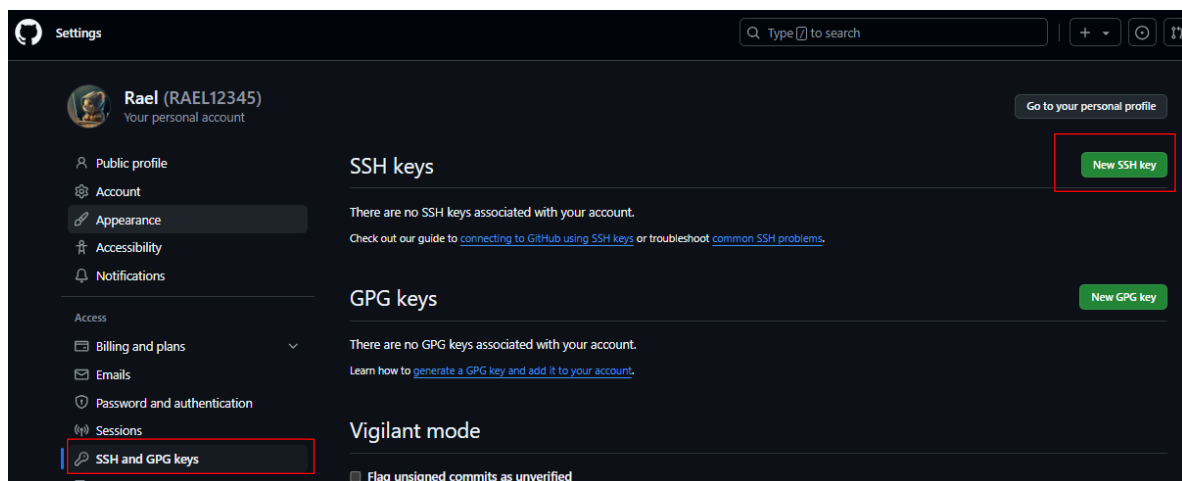
4. Añadir la clave pública a tu cuenta de GitHub

- Iniciar sesión en GitHub:
- ir a GitHub y acceder a nuestra cuenta.
- Ir a la configuración de SSH:

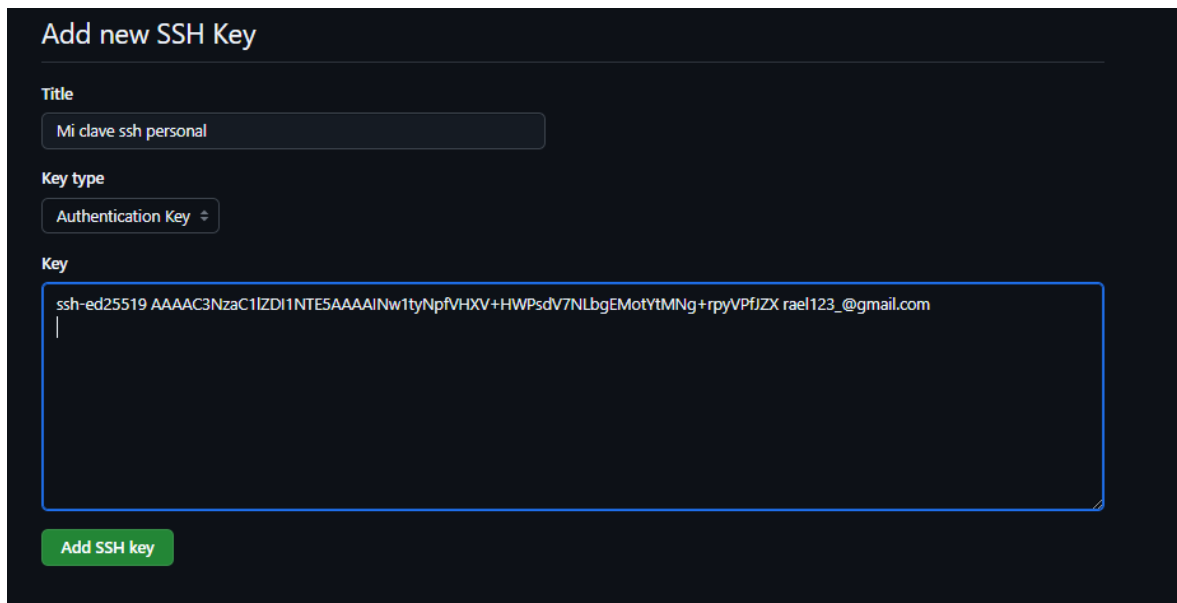
En la esquina superior derecha de la página, hacemos clic en la foto de perfil y seleccionamos "Settings".

En el menú de la izquierda, hacemos clic en "SSH and GPG keys".

Hacemos clic en el botón "New SSH key".



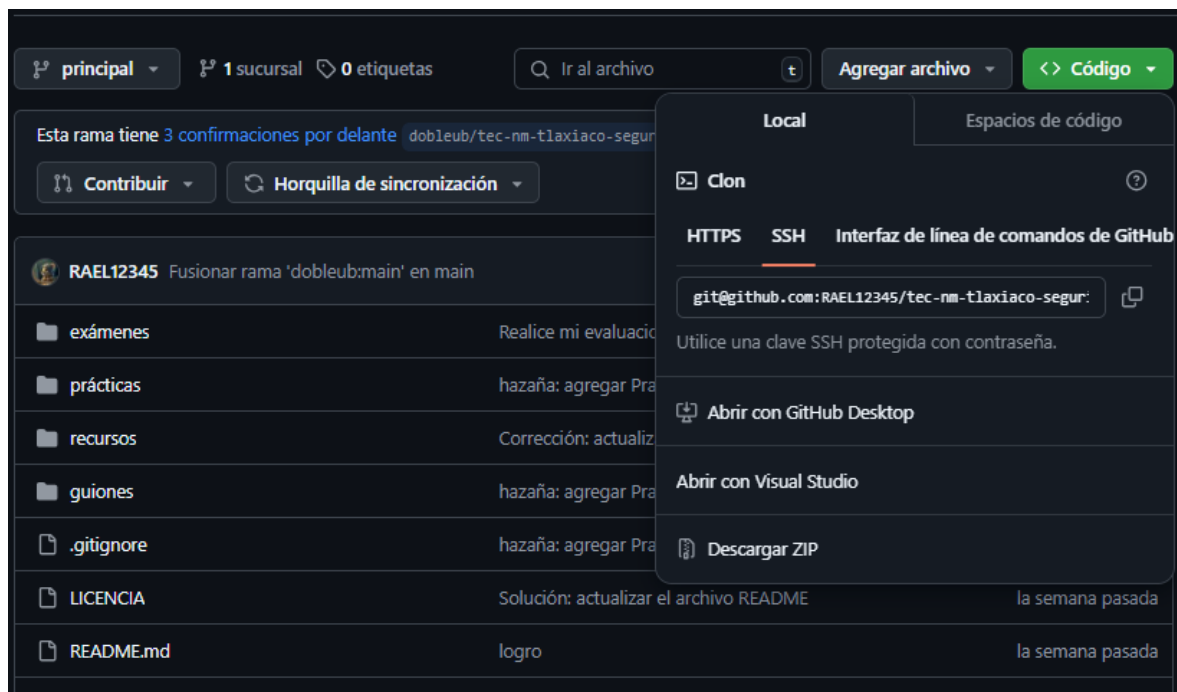
Pegamos la llave publica SSH y lo agregamos a nuestra cuenta de GitHub.



The screenshot shows the 'Add new SSH Key' interface in GitHub. It has a dark theme. The 'Title' field contains 'Mi clave ssh personal'. The 'Key type' dropdown is set to 'Authentication Key'. The 'Key' text area contains the SSH public key: `ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAINwtYnpfVHXV+HWPsdv7NLbgEMotYtMNg+rpYVPfJZX rael123_@gmail.com`. At the bottom is a green 'Add SSH key' button.

5. Realizar un clon con SSH

- Nos vamos a algún repositorio que tengamos en nuestra cuenta de GitHub y copiamos la url del SSH.



The screenshot shows a GitHub repository page for 'RAEL12345'. The repository has 1 branch and 0 tags. A dropdown menu is open for cloning the repository. The 'SSH' tab is selected, showing the URL: `git@github.com:RAEL12345/tec-nm-tlaxiaco-segur:`. Below the URL, it says 'Utilice una clave SSH protegida con contraseña.' Other options in the menu include 'Abrir con GitHub Desktop', 'Abrir con Visual Studio', and 'Descargar ZIP'. The repository file list on the left includes folders like 'exámenes', 'prácticas', 'recursos', 'guiones' and files like '.gitignore', 'LICENCIA', and 'README.md'.

- Ingresamos a la carpeta local en donde deseamos clonar el repositorio e ingresamos el comando git clone "url del SS"

La primera vez que intentamos conectarnos un nuevo host a través de SSH nos aparecerá el siguiente mensaje, y nos advierte que la autenticidad del servidor remoto no ha sido verificada antes en nuestra máquina. Si confías en el servidor, puedes escribir "yes" para continuar, y la clave se almacenará en el archivo ~/.ssh/known_hosts para futuras conexiones.

```
Rael@DESKTOP-M7HMA9A MINGW64 ~/Desktop/Seguridad y Virtualizacion
$ git clone git@github.com:RAEL12345/tec-nm-tlaxiaco-seguridad-virtualizacion.git
Cloning into 'tec-nm-tlaxiaco-seguridad-virtualizacion'...
The authenticity of host 'github.com (140.82.112.4)' can't be established.
ED25519 key fingerprint is SHA256:+DiY3wvV6TuJJhbpZisF/zLDA0zPMSvHdkr4UvCOqU.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
```

- Una vez confirmado ingresamos la contraseña de nuestro SSH y clonara el repositorio:

```
Warning: Permanently added 'github.com' (ED25519) to the list of known hosts.
Enter passphrase for key '/c/Users/Dell/.ssh/id_ed25519':
remote: Enumerating objects: 63, done.
remote: Counting objects: 100% (21/21), done.
remote: Compressing objects: 100% (16/16), done.
remote: Total 63 (delta 8), reused 16 (delta 5), pack-reused 42 (from 1)
Receiving objects: 100% (63/63), 27.92 MiB | 294.00 KiB/s, done.
Resolving deltas: 100% (18/18), done.

Rael@DESKTOP-M7HMA9A MINGW64 ~/Desktop/Seguridad y Virtualizacion
$ |
```

4.- Crea un certificado SSL auto firmado con una validez de 365 días y añádelo a un servidor web local. Realice una petición GET al servidor web local utilizando curl muestra el certificado SSL.

SSL (Secure Sockets Layer) es un protocolo de seguridad que se utiliza para establecer una conexión segura y cifrada entre un cliente (como un navegador web) y un servidor (como un sitio web). Aunque hoy en día se utiliza principalmente su sucesor, TLS (Transport Layer Security), el término SSL sigue siendo ampliamente utilizado para referirse a esta tecnología.

Un certificado SSL auto firmado es un certificado que no está emitido por una autoridad certificadora (CA), sino que es generado y firmado por el propio creador del certificado.

Generación del Certificado SSL Auto Firmado:

1. Tener instalado openSSI
2. Abrir el cmd de nuestro equipo y nos dirigimos a la ruta de instalación del openSSI .

```
C:\> Símbolo del sistema
Microsoft Windows [Versión 10.0.19045.4780]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Dell>cd C:\Program Files\OpenSSL-Win64\bin

C:\Program Files\OpenSSL-Win64\bin>
```

- ### 3. Generar la clave privada:

- Ejecutamos el siguiente comando para generar una clave privada.

[illegible]

Se utiliza la opción `nodes` si queremos que nuestra llave privada no requiera una contraseña de otro modo la removeremos.

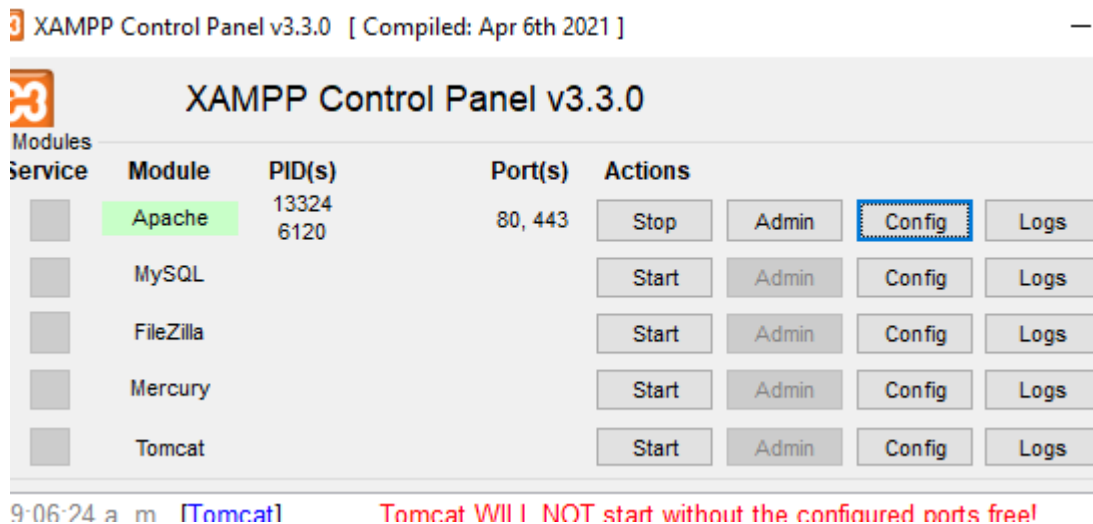
Deberemos sustituir `server.key` por el nombre que le queramos poner a nuestra llave privada, y `server.cer` por el nombre que deseemos asignar a nuestro certificado.

Con el comando `days` indicaremos cuantos días será válido nuestro certificado, pueden poner la cantidad de días que deseen, en el ejemplo he puesto un año, es decir 365 días.

Y por último y de manera opcional pueden usar la opción `subj` para indicar el dominio de nuestro certificado sin necesidad de llenar todo el cuestionario interactivo que aparece al ejecutar el comando.

4. Añadir el Certificado SSL al Servidor Web Local

Abrimos xampp para configurar el apache en el archivo “`httpd-ssl.conf`”



Añadimos al archivo lo siguiente reemplazando las rutas correctas a nuestro certificado y clave privada:

```
SSLCertificateFile "C:\Program Files\OpenSSL-Win64\bin\server.cer"
```

```
# Server Private Key:
# If the key is not combined with the certificate, use this
# directive to point at the key file. Keep in mind that if
# you've both a RSA and a DSA private key you can configure
# both in parallel (to also allow the use of DSA ciphers, etc.)
# ECC keys, when in use, can also be configured in parallel
SSLCertificateKeyFile "C:\Program Files\OpenSSL-Win64\bin\server.key"
```

Realizamos una petición GET al servidor web local utilizando curl desde la terminal en donde se obtiene la siguiente respuesta.

```
C:\Program Files\OpenSSL-Win64\bin>curl -vk https://localhost
* Host localhost:443 was resolved.
* IPv6: ::1
* IPv4: 127.0.0.1
* Trying [::1]:443...
* Connected to localhost (::1) port 443
* schannel: disabled automatic use of client certificate
* ALPN: curl offers http/1.1
* ALPN: server accepted http/1.1
* using HTTP/1.x
> GET / HTTP/1.1
> Host: localhost
> User-Agent: curl/8.7.1
> Accept: */*
>
* Request completely sent off
< HTTP/1.1 200 OK
< Date: Fri, 30 Aug 2024 17:39:54 GMT
< Server: Apache/2.4.54 (Win64) OpenSSL/1.1.1p PHP/8.2.0
< Content-Length: 2925
< Content-Type: text/html; charset=UTF-8
<
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<html>
  <head>
    <title>Index of /</title>
  </head>
  <body>
<h1>Index of /</h1>
    <table>
      <tr><th valign="top"></th><th><a href="?C=N;O=D">Name</a></th><th><a href="
0-A">Size</a></th><th><a href="?C=D;O=A">Description</a></th></tr>
      <tr><th colspan="5"><hr></th></tr>
      <tr><td valign="top"></td><td><a href="Proyecto-copia/">Proyecto-copia/</a>
d><td align="right"> - </td><td>&nbsp;</td></tr>
      <tr><td valign="top"></td><td><a href="Proyecto/">Proyecto/</a>
align="right"> - </td><td>&nbsp;</td></tr>
      <tr><td valign="top"></td><td><a href="Proyecto%20PHP/">Proyecto PHP/</a>
d><td align="right"> - </td><td>&nbsp;</td></tr>
      <tr><td valign="top"></td><td><a href="applications.html">applications.html</a>
d><td align="right">3.5K</td><td>&nbsp;</td></tr>
      <tr><td valign="top"></td><td><a href="bitnami.css">bitnami.css</a>
```

Conclusión

Seguridad y Validación de Contraseñas: Al crear este programa que, valida la seguridad de una contraseña, aprendimos cómo implementar y aplicar las mejores prácticas de seguridad recomendadas, como la verificación de longitud, caracteres especiales, y patrones de repetición. Esto nos dio la noción de cómo construir algoritmos para validar entradas del usuario de manera efectiva y segura.

Generación de Contraseñas Seguras: Al crear un generador de contraseñas seguras nos permitió comprender mejor la importancia de la aleatoriedad en la seguridad. Aprendimos cómo combinar diferentes tipos de caracteres para crear contraseñas robustas que cumplan con los criterios de seguridad, y cómo evitar patrones previsibles que podrían comprometer la seguridad.

Gestión de Claves SSH y GitHub: Al crear y utilizar un certificado SSH para autenticarte en GitHub, comprendimos cómo funcionan las claves públicas y privadas, y cómo se utilizan para garantizar una conexión segura. Esto es esencial

para la gestión de repositorios remotos de manera segura, especialmente cuando se trabaja en proyectos colaborativos o en entornos de producción.

Implementación y Uso de Certificados SSL: Crear un certificado SSL autofirmado y añadirlo a un servidor web local comprendimos los conceptos básicos de criptografía y la importancia de las conexiones seguras en la web. Al realizar una petición GET utilizando curl y visualizar el certificado SSL, entendimos cómo se establecen y validan las conexiones HTTPS seguras, lo cual es fundamental para proteger la transmisión de datos en la web.