Universidad Alfonso X El Sabio

Grado en Ingeniería Matemática

Realización de la fase de reconocimiento o footprinting de la Web de la UAX

Criptografía y Seguridad

Autores:

Carlota Martín-Anero
Alejandro Balduz López
María Pérez-Serrabona

Mayo 2023

Índice

1. Introducción

La fase de reconocimiento o footprinting es uno de los primeros pasos en una prueba de penetración o test de intrusión. Esta fase consiste en recopilar información sobre el objetivo de la prueba, que puede ser una red, una aplicación o un sitio web, entre otros. El objetivo del footprinting es identificar debilidades y vulnerabilidades en el sistema objetivo, con el fin de evaluar la seguridad y proteger la información crítica de la organización.

El footprinting se lleva a cabo a través de técnicas de exploración pasivas, que buscan recopilar información de fuentes públicas disponibles en la red, como sitios web, foros, redes sociales, entre otros. Estas técnicas no implican una interacción directa con el sistema objetivo, lo que hace que sean menos intrusivas y por lo tanto menos susceptibles de ser detectadas.

El footprinting es una parte fundamental de cualquier prueba de penetración y su importancia radica en la posibilidad de identificar los puntos débiles de un sistema y prevenir posibles ataques informáticos. En la actualidad, con el creciente uso de la tecnología y la constante evolución de las amenazas informáticas, el footprinting se ha vuelto más relevante que nunca. La recopilación de información y la identificación de vulnerabilidades es esencial para garantizar la seguridad de los sistemas informáticos y proteger la información sensible y confidencial de las organizaciones.

Figura 1: Footprinting

Una de las herramientas más utilizadas en la fase de footprinting es Maltego, que permite recopilar y visualizar información de múltiples fuentes de datos. Maltego es una herramienta de código abierto que utiliza una variedad de técnicas y algoritmos para identificar relaciones entre diferentes entidades y presentarlas de manera clara y visual.

Además, otra técnica utilizada en la fase de reconocimiento es el web scraping, que implica la extracción de información de una página web mediante la automatización de un proceso de navegación y extracción de datos. Python es uno de los lenguajes de programación más utilizados para el web scraping debido a su facilidad de uso y a la gran cantidad de bibliotecas disponibles.

En este trabajo se llevará a cabo la fase de reconocimiento o footprinting de la página web de la Universidad Alfonso X .El Sabio", utilizando técnicas de exploración pasivas y herramientas como Maltego y Python para obtener información sobre la estructura, arquitectura y posibles vulnerabilidades del sitio. Con los resultados obtenidos, se evaluará la seguridad del sitio y se propondrán recomendaciones para mejorar la protección de la información crítica de la universidad.

Es importante destacar que la realización de una fase de reconocimiento adecuada puede ahorrar tiempo y recursos en el proceso de una prueba de penetración, ya que permite identificar posibles vulnerabilidades y riesgos antes de realizar ataques más avanzados. Asimismo, esta fase es esencial para cualquier persona o empresa que busque mejorar la seguridad de su sitio web y prevenir posibles ataques informáticos.

Por lo tanto, el uso de herramientas como Maltego y Python para la fase de reconocimiento y web scraping puede proporcionar una gran cantidad de información valiosa para la evaluación de la seguridad de un sitio web. En este trabajo, se utilizarán estas herramientas para recopilar información de la página web de la Universidad Alfonso X .El Sabioz evaluar su seguridad, lo que puede ayudar a la universidad a tomar medidas para proteger su información crítica y mejorar su seguridad en línea.

1.1. Objetivos

El objetivo principal de este trabajo es realizar una prueba de penetración o test de intrusión en la página web de la Universidad Alfonso X .El Sabio". Para lograr este objetivo, se llevará a cabo la fase de reconocimiento o footprinting del sitio, con el fin de recopilar información sobre la estructura, arquitectura y posibles vulnerabilidades del mismo.

En primer lugar, se realizará un estudio detallado de las herramientas más utilizadas en la fase de reconocimiento, con especial atención en Maltego, una herramienta de código abierto que permite recopilar y visualizar información de múltiples fuentes de datos. Además, se utilizará Python, un lenguaje de programación muy popular, para llevar a cabo el proceso de web scraping y obtener información adicional del sitio.

Una vez se tenga la información necesaria, se procederá a analizarla y evaluar la seguridad del sitio. Se identificarán los puntos débiles y las posibles vulnerabilidades, con el objetivo de proponer recomendaciones para mejorar la protección de la información crítica de la universidad.

Cabe aclarar que todo este proceso es con el permiso de la universidad, para realizar una práctica con un único fin educativo para la asignatura de **Criptografía y seguridad**.

2. Web Scraping con Python

En esta presente sección, se procederá a realizar un estudio de una página web, mediante el lenguaje de programación Python.Como ya hemos visto durante otros trabajos, Python es una poderosa herramienta que puede ser empleada para realizar infinidad de acciones como el modelado de los datos, automatización de procesos o lo que trataremos en esta sección, *web scraping*.

Para la realización de esta práctica, se ha hecho uso de la página web de la universidad privada de Madrid, Alfonso X El Sabio, que tiene por url *https://www.uax.com*. El objetivo de esta sección, será obtener información a cerca de su página web, así como de su contenido, sin necesidad de entrar a ella mediante la red de Internet.

Para la realización de esta práctica, se ha hecho uso de la librería *Beautiful Soup*, esta es, una biblioteca de análisis de datos en Python que se utiliza para extraer información de páginas web. Permite analizar y manipular fácilmente el HTML y el XML de una página web. Teniendo acceso a estas partes de la web, podemos ver de forma fácil la naturaleza de la misma así como su contenido.
A lo largo de esta parte, se comentará el código empleado así como sus funciones y utilidad.

2.1. Primer Scrapeo

Lo primero que debemos hacer como resulta obvio, es seleccionar la página sobre la que vamos a trabajar mediante *request*. Con el uso de esta librería junto con el paquete Beautiful soup, conseguimos visualizar todo el código HTML asociado a la página en cuestión.

```
1    import requests
2
3    req = requests.get("https://www.uax.com")
4    print(req.text)
```

Ahora, accederemos al DOM de la página mediante el uso de esta librería.

```
1    from bs4 import BeautifulSoup
2
3    soup = BeautifulSoup(req.text)
4    print(soup)
```

Aunque el output sea relativamente parecido, con estos últimos comandos, lo que se permite es acceder a los elementos que componen el código, como son las clases, identificadores o etiquetas. Así pues, si queremos acceder al título de la página, emplearemos un comando como el que sigue:

```
1    soup.select("title")
```

Además de poder filtrar por las etiquetas, podemos seleccionar únicamente uno de los elementos que queramos estudiar, pues actúa como una lista corriente de python. Si quisiéramos acceder al texto del primer título:

```
1    soup.select("title")[0].getText()
```

De donde obtendríamos la siguiente salida: *'Universidad Alfonso X El Sabio: Universidad Privada en Madrid - UAX'*, como era de esperar.

Del mismo modo se pueden filtrar por otro tipo de etiquetas como los *h1, a, p....* No solo eso, si no que además, podemos tener acceso a los tributos que definen cada una de las etiquetas. Supongamos que queremos obtener la referencia asociada a una etiqueta de enlace, podríamos expresar:

```
1    a=soup.select("a")[0]
2    a['href']
```

Obteniendo entonces la URL asociada a la primera etiqueta de tipo *a* que se encuentre en el HTML de nuestra página de estudio.

La siguiente pregunta natural que nos surge es, la posibilidad de conseguir todas los atributos asociados a un cierto elemento del código. Para ello ejecutamos:

```
a=soup.select("a")[0]
a.attr.intems()
```

Mediante el diccionario *attr*, lo que conseguimos es un mapeo de todos loa atributos asociados a un elemento, así como sus valores. Generalizando esta última parte, también es posible conseguir todos los atributos asociados a una etiqueta general, así como sus valores.

```
for meta in soup.select("meta"):
    for atributo, valor in meta.attrs.items():
        print(f"{atributo}: {valor}")
```

De esta forma, lo que se consigue es un gran diccionario con todos los atributos de todos los elementos de tipo *meta*, así como sus valores en la página de estudio.

Después de este primer scrapeo, podemos intuir la fuerza que tiene esta librería, pues en simples comandos, podemos acceder a información muy poderosa de una cierta página web sin la necesidad de entrar en ella. Con Beautiful Soup, conseguimos la información de forma precisa y rápida, lista para ser empleada con otras herramientas.

2.2. Uso de selectores

En esta sección, seguiremos trabajando con la página web de la universidad salvo que esta vez, lo haremos poniendo en práctica algunos selectores.

Los selectores son elementos utilizados en programación para seleccionar un conjunto específico de datos o elementos de una estructura de datos más grande. En términos más simples, los selectores son herramientas que permiten a los programadores especificar un subconjunto de elementos en una colección de datos. En este caso, lo haremos desde CSS. En este lenguaje, los selectores se utilizan para seleccionar elementos HTML específicos y aplicar estilos a ellos. Los selectores CSS pueden ser basados en diferentes características del elemento, como el tipo de etiqueta, la clase, el ID, la posición, el estado, etc.

Como bien hemos dicho en la sección anterior, lo primero que debemos de hacer es realizar una solicitud(request) a la página web *https://uax.com* utilizando la biblioteca de Python 'requests'. A continuación, la respuesta obtenida se pasa a la función 'BeautifulSoup' de la biblioteca 'bs4' para analizar el contenido HTML de la página. Este proceso es el mismo durante todo el scrapeo.

Luego, se selecciona el elemento 'title' de la página web utilizando el método 'select' de la instancia 'soup'. Esto devuelve una lista de todos los elementos de la página que coinciden con el selector especificado. En este caso, se espera que haya solo un elemento 'title' en la página, por lo que se accede al primer elemento de la lista (índice [0]).

```
1  import requests
2  from bs4 import BeautifulSoup
3
4  req = requests.get("https://uax.com")
5  soup = BeautifulSoup(req.text)
6
7  title = soup.select("title")[0].getText()
8  print(title)
```

Así es como conseguimos obtener el título de la página, haciendo uso de selectores.

El siguiente paso, a modo de ejemplo también, es scrapear el índice. En este caso, es parecido al anterior. Lo que se hará es seleccionar un elemento específico de la página web utilizando su identificador para posteriormente iterar a través de todos los elementos 'a' que se encuentran dentro del elemento. De esta forma, estamos accediendo a todos los elementos del identificador mencionado. El resultado será todas la secciones que nos presenta la página web; Nuestra historia 25 años, UAX Rafa Nadal School of Sport, Innovación e Investigación, La Universidad de la Empresa, Portal de transparencia, Campus y residencias, Centros y laboratorios, Instalaciones deportivas, Campus Tour Interactivo, Acceso a grados y dobles grados, Acceso a másteres y postgrados, Acceso a ciclos formativos, Acceso mayores de 25 años, Acceso estudiantes online, Orientación Universitaria, Becas, ayudas y financiación, Beneficios por ser estudiante UAX.

```
1  toc = soup.select("#megamenu-1591")[0]
2
3  for a in toc.select("a"):
4      print(a.getText())
```

En esta parte, haciendo uso de este tipo de cosas podremos mostrar enlaces, formatear los espacios, dividir en categorías...

Por último, veremos cómo scrapear una imagen de la página web. En el ejemplo, se realiza una solicitud HTTP a la página web especificada por url y se analiza el contenido HTML de la respuesta utilizando BeautifulSoup al igual que antes. Luego, se busca la etiqueta 'img' utilizando la función

'find' de BeautifulSoup y se extrae la URL de la imagen utilizando el atributo 'src' de la etiqueta. Se utiliza el atributo 'class' de la etiqueta para asegurarse de que se está obteniendo la imagen del logotipo de la UAX. Finalmente, se realiza una segunda solicitud HTTP para descargar la imagen y se guarda en un archivo llamado 'uax logo.png' utilizando un bloque 'with open'.

El código implementado para scrapear la imagen, nos saltaba un error que es bastante común que nos decía: 'NoneType' object is not subscriptable. Este error ocurre cuando intentas acceder a un elemento o propiedad de un objeto 'None' en Python. En el contexto del web scraping, puede ocurrir cuando BeautifulSoup no encuentra un elemento en la página web que estás intentando acceder. Para solucionar este error, podemos agregar una comprobación para asegurarnos de que el objeto que estamos intentando acceder no sea 'None' antes de intentar acceder a sus propiedades. En este caso, podemos verificar que la etiqueta 'img' que estamos buscando existe en la página web antes de intentar acceder a su atributo 'src'. En el código referente a esta sección se entenderá de una mejor forma lo que se está haciendo.

```
import requests
from bs4 import BeautifulSoup

url = "https://www.uax.com/"

# Realizar solicitud HTTP
response = requests.get(url)

# Analizar contenido HTML de la respuesta
soup = BeautifulSoup(response.text, "html.parser")

# Encontrar la etiqueta de imagen y obtener su URL
img_tag = soup.find("img", {"class": "img-responsive"})
if img_tag is not None:
    img_url = img_tag.get("src")
    if img_url is not None:
        # Descargar la imagen y guardarla en un archivo
        response = requests.get(img_url)
        with open("uax_logo.png", "wb") as f:
            f.write(response.content)
    else:
        print("La etiqueta img no tiene el atributo src.")
else:
    print("No se pudo encontrar la etiqueta img.")
```

2.3. Scraping automatizado

En esta sección, se intenta programar un script que sea capaz de scrapear la página web de la universidad automáticamente. Sabemos qué estructura tiene la web, por tanto trataremos de iterar sobre titulaciones, facultades, contenido, cursos...

Lo primero que haremos será como siempre utilizar la biblioteca 'requests' para enviar una solicitud HTTP GET a la página web y obtener su contenido HTML. Luego, se utiliza la biblioteca BeautifulSoup para analizar el contenido HTML y extraer información. Después, buscaremos todas las etiquetas 'div' con la clase 'titulaciones' en la página web de la UAX, y para cada una de ellas extraeremos el contenido de la titulación, la modalidad y las etiquetas asociadas. Luego, se imprime la información de cada titulación en la consola.

```
import requests
from bs4 import BeautifulSoup

# Realizar solicitud HTTP GET a la pagina web de la UAX
url = 'https://www.uax.com/'
response = requests.get(url)

# Analizar contenido HTML de la respuesta
soup = BeautifulSoup(response.content, 'html.parser')

# Encontrar las titulaciones con su respectivo autor y etiquetas
titulaciones = soup.select('div.titulaciones')
for titulacion in titulaciones:
    # Extraer el contenido de la titulacion
    contenido = titulacion.select_one('span.contenido').text

    # Extraer el autor de la titulacion
    Modalidad = titulacion.select_one('small.Modalidad').text

    # Extraer las etiquetas de la titulacion
    etiquetas = [tag.text for tag in titulacion.select('div.facultad a.tag')]

    # Imprimir la informacion de la titulacion
    print(f'Titulacion: {contenido}')
    print(f'Modalidad: {Modalidad}')
    print(f'Etiquetas: {', '.join(etiquetas)}')
    print('\n')
```

Ahora que ya tenemos este código, podríamos adaptar este código a una función que a partir de una porción de la URL almacene mediante diccionarios las Titulaciones.

```

def scrape_uax_titulaciones(porcion_url):
    # Completar la URL de la pagina web de la UAX con la porcion URL dada
    url = ftitulaciones'https://www.uax.com/{porcion_url}titulaciones'

    # Realizar solicitud HTTP GET a la pagina web de la UAX
    response = requests.get(url)

    # Analizar contenido HTML de la respuesta
    soup = BeautifulSoup(response.content, titulaciones'html.parsertitulaciones')

    # Inicializar diccionario para almacenar informacion de las titulaciones
    titulaciones_dict = {}

    # Encontrar las titulaciones con su respectivo autor y etiquetas
    titulaciones = soup.select(titulaciones'div.quotetitulaciones')
    for i, titulacion in enumerate(titulaciones):
        # Extraer el contenido de la titulacion
        contenido = titulacion.select_one(titulaciones'span.contenidotitulaciones'
            ).text
```

```
20
21            # Extraer el autor de la titulacion
22            modalidad = titulacion.select_one(titulaciones'small.Modalidadtitulaciones
                  ').text
23
24            # Extraer las etiquetas de la titulacion
25            etiquetas = [tag.text for tag in titulacion.select(titulaciones'div.
                  facultad_a.tagtitulaciones')]
26
27            # Almacenar la informacion de la titulacion en el diccionario
28            titulaciones_dict[ftitulaciones'Titulacion_{i+1}titulaciones'] = {
                  titulaciones'contenidotitulaciones': contenido, titulaciones'
                  modalidadtitulaciones': modalidad, titulaciones'etiquetastitulaciones':
                   etiquetas}
29
30        # Devolver el diccionario con la informacion de las titulaciones
31        return titulaciones_dict
32
33    titulaciones = scrape_uax_titulaciones(titulaciones'estudios-oficiales/
          titulaciones')
34    print(titulaciones)
```

La función toma una porción de la URL de la página web de la UAX como argumento, completa la URL y realiza una solicitud HTTP GET para obtener el contenido HTML de la página. Luego, utiliza BeautifulSoup para analizar el contenido HTML y extraer la información de las titulaciones.

La función almacena la información de cada titulación en un diccionario, donde la clave es el número de la titulación y el valor es otro diccionario que contiene el contenido de la titulación, la modalidad y las etiquetas. Finalmente, la función devuelve el diccionario con la información de las titulaciones.

La clave es utilizar nuestra función de forma recursiva detectando si la página tiene el enlace 'Next' y cargando la siguiente página de manera que podamos. Podemos integrar este concepto en nuestra función 'scrap uax titulaciones' para devolver no solo las Titulaciones de la página, sino también si hay una página siguiente:

```
1     def scrape_uax_titulaciones(porcion_url, titulaciones_dict=None):
2         # Completar la URL de la pagina web de la UAX con la porcion URL dada
3         url = ftitulaciones'https://www.uax.com/{porcion_url}titulaciones'
4
5         # Realizar solicitud HTTP GET a la pagina web de la UAX
6         response = requests.get(url)
7
8         # Analizar contenido HTML de la respuesta
9         soup = BeautifulSoup(response.content, titulaciones'html.parsertitulaciones')
10
11        # Si es la primera pagina, inicializar diccionario para almacenar informacion
              de las titulaciones
12        if titulaciones_dict is None:
13            titulaciones_dict = {}
14
15        # Encontrar las titulaciones con su respectivo autor y etiquetas
16        titulaciones = soup.select(titulaciones'div.quotetitulaciones')
17        for i, titulacion in enumerate(titulaciones):
18            # Extraer el contenido de la titulacion
19            contenido = titulacion.select_one(titulaciones'span.contenidotitulaciones'
                  ).text
20
21            # Extraer el autor de la titulacion
22            modalidad = titulacion.select_one(titulaciones'small.Modalidadtitulaciones
                  ').text
23
24            # Extraer las etiquetas de la titulacion
25            etiquetas = [tag.text for tag in titulacion.select(titulaciones'div.
```

```
            facultad_a.tagtitulaciones')]
26
27          # Almacenar la informacion de la titulacion en el diccionario
28          titulaciones_dict[ftitulaciones'Titulacion_{len(titulaciones_dict)+1}
                titulaciones'] = {titulaciones'contenidotitulaciones': contenido,
                titulaciones'modalidadtitulaciones': modalidad, titulaciones'
                etiquetastitulaciones': etiquetas}
29
30      # Buscar boton titulaciones'Nexttitulaciones' en la pagina actual
31      next_button = soup.select_one(titulaciones'.next.page-numberstitulaciones')
32
33      # Si no hay boton titulaciones'Nexttitulaciones', detener la recursion y
            devolver el diccionario de las titulaciones
34      if not next_button:
35          return titulaciones_dict
36
37      # Si hay boton titulaciones'Nexttitulaciones', obtener el enlace y llamar
            recursivamente la funcion con la nueva URL
38      next_link = next_button[titulaciones'hreftitulaciones']
39      return scrape_uax_titulaciones(next_link, titulaciones_dict)
```

La función ahora toma un segundo argumento opcional, 'titulaciones dict', que es el diccionario de las titulaciones que se está construyendo. Si este argumento no se proporciona, se crea un nuevo diccionario.

La función sigue buscando las titulaciones en la página actual y las agrega al diccionario. Luego, busca el botón 'Next' en la página actual y verifica si está presente. Si no hay un botón 'Next', la función detiene la recursión y devuelve el diccionario de las titulaciones. Si hay un botón 'Next', la función obtiene su enlace y llama recursivamente la función con la nueva URL y el diccionario actualizado.

Ahora, vamos a implementar una función que scrapee todas las páginas mientras haya una siguente o, alternativamente, podemos establecer un límite para optimizar el proceso y no saturar al servidor.

```
1   def scrape_website(url, limit=None, verbose=False):
2       page_number = 1
3       while True:
4           # Hacemos la peticion HTTP y parseamos el HTML con BeautifulSoup
5           req = requests.get(url.format(page_number))
6           soup = BeautifulSoup(req.content, 'html.parser')
7
8           # Buscamos las Titulaciones y las imprimimos
9           titulaciones = soup.select('div.quote')
10          for titulacion in titulaciones:
11              data = {}
12              data['titulo'] = titulacion.select_one('span.contenido').text
13              data['autor'] = titulacion.select_one('small.Modalidad').text
14
15              etiquetas = titulacion.select('div.facultad_a.tag')
16              data['etiquetas'] = [etiqueta.text for etiqueta in etiquetas]
17
18              print(data)
19
20          if verbose:
21              print(f'Extrayendo_pagina_{page_number}')
22
23          # Si hemos llegado al limite, salimos del loop
24          if limit is not None and page_number >= limit:
25              break
26
27          # Buscamos el enlace de la siguiente pagina
28          next_page = soup.select_one('a.next')
29          if next_page is None:
30              break
```

```
31
32              # Actualizamos la URL para la siguiente pagina
33              url = next_page['href']
34              page_number += 1
35
36      scrape_website('https://www.uax.com/page/{}/', limit=5, verbose=True)
```

Hemos definido una función que reciba la URL de la página inicial, un límite opcional y un parámetro para indicar si queremos imprimir el progreso de la extracción. Estamos indicando que queremos extraer hasta 5 páginas y que se imprima el progreso de la extracción en cada iteración.

Seguidamente, implementaremos la clase Titulaciones y el método scrapear siguiendo lo que hemos realizado anteriormente. Solo generaremos el fichero si ejecutamos el método scrapear, los demás métodos 'lista', 'etiqueta' y 'modalidad' analizarán el contenido del fichero volcado en la memoria, pero nunca scrapearán nada directamente.

```
1   import csv
2
3   class Titulaciones:
4
5       # Variable de clase para almacenar las Titulaciones en la memoria
6       titulaciones = []
7
8       @staticmethod
9       def scrapear():
10          # Scrapeamos todas las Titulaciones, ponemos un limite pequenio para hacer
                pruebas
11          titulaciones = scrape_website("https://www.uax.com/page/{}/", limit=5,
                verbose=True)
12          if titulaciones is not None:
13              Titulaciones.titulaciones = titulaciones
14              # Guardamos las Titulaciones scrapeadas en un fichero CSV volcandolas
                    de la lista de dicts
15              with open("titulaciones.csv", "w") as file:
16                  # Definimos el objeto para escribir con las cabeceras de los
                        campos
17                  writer = csv.DictWriter(file, fieldnames=["contenido", "Modalidad"
                        , "facultad"])
18                  # Escribimos las cabeceras
19                  writer.writeheader()
20                  # Escribimos cada cita en la memoria en el fichero
21                  for quote in Titulaciones.titulaciones:
22                      writer.writerow(quote)
23          else:
24              print("La lista de titulaciones es None")
25
26  Titulaciones.scrapear()
```

En este punto del scrapeo, se debería tener un conjunto de datos de tipo csv con todas las titulaciones y su información (Facultad, Modalidad y Contenido). Lo que se va a proponer ahora, es una clase que lea este mismo fichero con las titulaciones y listar una a una todas ellas, con su contenido relacionada.

Este código se basa en la lectura de un conjunto de datos con una posterior escritura en el mismo, para por último, listar todas las titulaciones encontradas. A continuación se muestra el código, con comentarios que facilitan la comprensión del mismo.

```
1   import os
2   import csv
3   #Clase que devuelve todas las titulaciones de la universiada
4   class Titulaciones:
5
```

```
6         # Variable de clase para almacenar las titulaciones en la memoria
7         titulaciones = []
8
9         # si existe el fichero, lo leemos con la funcion  csv.DictReader
10        if os.path.exists("titulaciones.csv"):
11            with open("titulaciones.csv", "r") as file:
12                data = csv.DictReader(file)
13                for titulacion in data:
14                    # La lista es una cadena, hay que reevaluarla
15                    titulaciones['Facultad'] = eval(titulaciones['Facultad'])
16                    titulaciones.append(titulacion)
17
18        @staticmethod
19        def scrapear():
20            # Scrapeamos todas las titulaciones, ponemos un limite pequenio para hacer
                  pruebas
21            Titulaciones.titulaciones = scrap_site(limit=2)
22
23            # Guardamos las titulaciones encontradas en un fichero CSV volcandolas de
                  la lista de dicts
24
25            #Sobre el mismo fichero , se escribe con la funcion csv.DictWriter, la
                  informacion
26            #relativa de cada una de las titulaciones
27
28            with open("titulaciones.csv", "w") as file:
29
30                # Definimos el objeto para escribir con las cabeceras de los campos
31                writer = csv.DictWriter(file, fieldnames=["Contenido", "Modalidad", "
                      Facultad"])
32
33                # Escribimos las cabeceras
34                writer.writeheader()
35
36                # Escribimos cada titulacion en la memoria en el fichero
37                for titulacion in Titulaciones.titulaciones:
38                    writer.writerow(titulacion)
39        #Por ultimo, funcion que lista las titulaciones
40        @staticmethod
41        def listar(limite=10):
42
43            for titulacion in Titulaciones.titulaciones[:limite]:
44                #muestra la informacion de cada una de las titulaciones
45
46                print(titulacion["Contenido"])
47                print(titulacion["Modalidad"])
48                for facultad in titulacion["Facultad"]:
49
50                    print(facultad, end="␣")
51
52                print("\n")
53
54    Titulaciones.listar(5)
```

Una vez tenemos una lista con todas las titulaciones que ofrece la universidad, se procede a crear un código capaz de filtrarlas por Facultad y Modalidad. De forma general, tendremos que recorrer la lista anterior, y comprobar cuales cumplen con los requisitos, para poder mostrarlas por pantalla. De igual modo, en el código se explica la funcionalidad de cada línea del mismo. Así pues:

```
1   import os
2   import csv
3
```

```
4   class Titulaciones:
5
6       #En primer parte del codigo, se conserva lo que veiamos antes
7
8       # Variable de clase para almacenar las titulaciones en la memoria
9       titulaciones = []
10
11      # Si existe el fichero, se procede a abrirlo y leerlo
12      if os.path.exists("titulaciones.csv"):
13          with open("titulaciones.csv", "r") as file:
14              data = csv.DictReader(file)
15              for titulaciones in data:
16                  # La lista es una cadena, hay que reevaluarla
17                  titulaciones['Facultad'] = eval(titulaciones['Facultad'])
18                  titulaciones.append(titulaciones)
19
20      #Sobreescribimos el fichero para poder almacenar la informacion de las
            titulaciones
21      @staticmethod
22      def scrapear():
23
24          # Scrapeamos todas las titulaciones, ponemos un limite pequenio para hacer
                pruebas
25
26          Titulaciones.titulaciones = scrap_site(limit=2)
27          # Guardamos las titulaciones scrapeadas en un fichero CSV volcandolas de
                la lista de dicts
28
29          with open("titulaciones.csv", "w") as file:
30              writer = csv.DictWriter(file, fieldnames=["Contenido", "Modalidad", "
                    Facultad"])
31              writer.writeheader()
32
33              #Guardamos al informacion de cada una de las titulaciones
34              for titulacion in Titulaciones.titulaciones:
35                  writer.writerow(titulacion)
36
37
38      #Listamos las titulaciones encontradas en la pagina
39      @staticmethod
40      def listar(limite=10):
41          #Mostramos la informacion de cada titulacion
42          for titulacion in Titulaciones.titulaciones[:limite]:
43              print(titulacion["Contenido"])
44              print(titulacion["Modalidad"])
45              for tag in titulacion["tags"]:
46                  print(tag, end="␣")
47              print("\n")
48
49
50      # Clasificar por facultad de la titulacion
51      @staticmethod
52      def Facultad(nombre=""):
53
54          #Se recorren las titulaciones
55          for titulacion in Titulaciones.titulaciones:
56
57              #Se comprueba si la Facultad coincide con la que se quiere buscar
58              if nombre in titulacion["Facultad"]:
59                  #Se muestra la informacion relativa a esta titulacion
60
61                  print(titulacion["Contenido"])
62                  print(titulacion["Modalidad"])
63                  for facultad in titulacion["Facultad"]:
64                      print(facultad, end="␣")
```

```
65                  print("\n")
66
67
68          #Seguimos un proceso muy similar para el filtrado por modalidad
69          @staticmethod
70          def Modalidad(nombre=""):
71
72              #Se recorren todas las titulaciones
73              for titulacion in Titulaciones.titulaciones:
74
75                  #Si la modaliadd concuerda con la buscada, se imprime por pantalla la
                        informacion de esta titulacion
76
77                  if nombre == titulacion["Modalidad"]:
78                      print(titulacion["Contenido"])
79                      print(titulacion["Modalidad"])
80                      for facultad in titulacion["Facultad"]:
81                          print(facultad, end="␣")
82                      print("\n")
83
84
85      #Se prueba el codigo anterior, intentando filtrar por aquellas titulaciones que se
            imparten en la Escual Politecnica
86      Titulaciones.Facultad("Politecnica")
87
88
89      #Comprobamos el codigo de nuevo, filtrando por aquellas titulaciones que se
            imparten con modalidad presencial.
90      Titulaciones.Modalidad("Presencial")
```

Como ya hemos visto, filtrar información de una web no es tan complicado, si se disponen de las herramientas necesarias para ello. En el código que acabamos de explicar, teníamos el número de páginas limitado, sin embargo, vamos a ver ahora el código necesario para poder hacer un scrapeo de toda la web y por tanto, conseguir un mayor número de información y más precisa.

Se procede a mostrar el código, comentando línea a línea lo que ocurre en él:

```
1   import os
2   import csv
3   import requests
4   from bs4 import BeautifulSoup
5
6
7   #Se accede a la p gina y se accede al DOM mediante BeautifulSoup
8   def scrap_quotes(url=""):
9       domain = "https://www.uax.com"
10      req = requests.get(f"{domain}{url}")
11      soup = BeautifulSoup(req.text)
12
13      #Se recorren las titulaciones y se consigue la informacion a traves de las
            etiquetas que forman el HTML
14
15      titulaciones = []
16      titulaciones_facultades = soup.select("div.Facultad")
17      for titulacion_facultad in titulaciones_facultades:
18          titulacion = {}
19          titulacion['Contenido'] = titulacion_facultad.select("span.Contenido")[0].
                getText()
20          titulacion['Modalidad'] = titulacion_facultad.select("small.modalidad")
                [0].getText()
21          titulacion['Facultad'] = []
22          for facultad in titulacion_facultad.select("div.facultad␣a.facultad"):
23              titulacion['Facultad'].append(facultad.getText())
24          quotes.append(titulacion)
25
26      next_url = None
```

```
27          link_tag = soup.select("li.next a")
28          if len(link_tag) > 0:
29              next_url = link_tag[0]['href']
30
31          print(f"P gina {domain}{url}, {len(titulacion)} titulaciones scrapeadas.")
32
33          return titulaciones, next_url
34
35
36      #Se devuelven todas las titulaciones encontradas
37      def scrap_site(limit=2):
38          todas_titulaciones = []
39          next_url = ""
40          while 1:
41              titulaciones, next_url = scrap_titulaciones(next_url)
42              todas_titulaciones += titulaciones
43              limit -= 1
44              if limit == 0 or next_url == None:
45                  return todas_titulaciones
46
47
48
49      #En esta clase, se realizan las mismas acciones que ve amos en el codigo justo
            anterior pero teniendo en cuenta todas las paginas de la web
50      class Titulaciones:
51          titulaciones = []
52
53          if os.path.exists("titulaciones.csv"):
54              with open("titulaciones.csv", "r") as file:
55                  data = csv.DictReader(file)
56                  for titulacion in data:
57                      titulacion['Facultad'] = eval(titulacion['Facultad'])
58                      titulacion.append(titulacion)
59
60          @staticmethod
61          def scrapear():
62              Titulaciones.titulaciones = scrap_site(limit=99) # <-- LIMITE MUY GRANDE
63              with open("titulaciones.csv", "w") as file:
64                  writer = csv.DictWriter(file, fieldnames=["Contenido", "Modalidad", "
                        Facultad"])
65                  writer.writeheader()
66                  for titulacion in Titulaciones.titulaciones:
67                      writer.writerow(titulacion)
68
69          @staticmethod
70          def listar(limite=10):
71              for titulacion in Titulaciones.titulacion[:limite]:
72                  print(titulacion["Contenido"])
73                  print(titulacion["Modalidad"])
74                  for facultad in titulaciones["Facultad"]:
75                      print(facultad, end=" ")
76                  print("\n")
77
78          @staticmethod
79          def Facultad(nombre=""):
80              for titulacion in Titulaciones.titulaciones:
81                  if nombre in titulaciones["Facultad"]:
82                      print(titulacion["Contenido"])
83                      print(titulacion["Modalidad"])
84                      for facultad in tiutlacion["Facultad"]:
85                          print(facultad, end=" ")
86                      print("\n")
87
88          @staticmethod
89          def Modalidad(nombre=""):
```

```
90            for titulacion in Titulaciones.titulaciones:
91                if nombre == titulacion["Modalidad"]:
92                    print(tiutlacion["Contenido"])
93                    print(titulacion["Modalidad"])
94                    for facultad in titulacion["Facultad"]:
95                        print(facultad, end="␣")
96                    print("\n")
97
98
99    #Probamos el c digo
100   Titulaciones.scrapear()
```

3. Maltego

3.1. Introducción

Maltego es una herramienta de inteligencia de código abierto utilizada para la investigación de amenazas y la recopilación de información. Se utiliza para recopilar información sobre una entidad en particular, como una persona, organización o sitio web, a partir de diferentes fuentes de información en línea y fuera de línea.

En nuestro proyecto, hemos utilizado Maltego para recopilar información sobre una organización objetivo. Para ello, hemos utilizado técnicas de OSINT para descubrir información relevante sobre la organización, como sus empleados, redes sociales, correos electrónicos y dominios relacionados.

Para realizar esta tarea, hemos utilizado la función de búsqueda de entidades de Maltego para buscar información relevante y luego hemos utilizado sus capacidades de análisis de gráficos para visualizar la información recopilada en un formato fácil de entender. De esta manera, hemos sido capaces de analizar la información de una manera más efectiva y tomar decisiones basadas en la inteligencia recopilada.

En resumen, Maltego es una herramienta esencial para la investigación de amenazas y la recopilación de información, y hemos utilizado sus capacidades para recopilar y analizar información sobre nuestra organización objetivo.

3.2. Domain

Vamos a realizar un *web scraping*, de la página de la ***Universidad Alfonso X el Sabio***, cuyo dominio web es: *www.uax.es*.

Para ello, vamos a seguir los siguientes pasos:

1. **Abrir Maltego**: Lo primero que debemos hacer es abrir la app de *Maltego*.
2. **Crear un nuevo gráfico**: una vez tengamos abierto la app de Maltego, creamos un nuevo gráfico. En este gráfico es donde vamos a agregar la información obtenida mediante el web scraping..

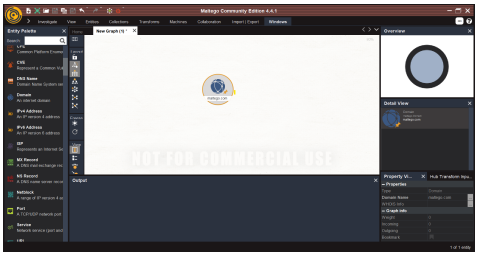


Figura 2: Nodo de dominio web

3. **Especificamos el dominio**: Una vez configurada la transformación en Maltego, debemos especificar el dominio en el que queremos realizar el web scraping. Para ello, podemos utilizar la herramienta .Entity Manager"de Maltego, en la que podemos crear una nueva entidad que represente el dominio y establecer las propiedades necesarias para realizar el web scraping correctamente. En uestro caso el dominio web que vamos a utilizar va a ser: *www.uax.es*.

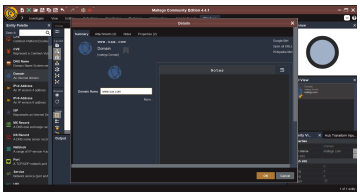


Figura 3: Escribimos nuestro dominio web

4. **Ejecutar la transformación**: Una vez que hemos configurado Maltego y especificado el dominio, podemos ejecutar la transformación para realizar el web scraping. Para ello, debemos seleccionar la entidad correspondiente al dominio y aplicar la transformación que hemos creado en el paso anterior. Esto generará un grafo en Maltego con la información obtenida del web scraping. En nuestro caso, hemos aplicado todas las transformaciones

Figura 4: Aplicamos las transformaciones

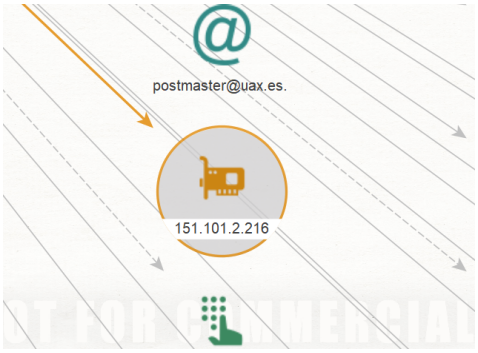
5. **Analizar los resultados**: Finalmente, debemos analizar los resultados obtenidos del web scraping en Maltego para extraer la información relevante para nuestro objetivo. Para ello, podemos utilizar las herramientas de análisis de Maltego para visualizar y explorar el grafo generado y extraer la información necesaria.

Figura 5: Grafo del dominio

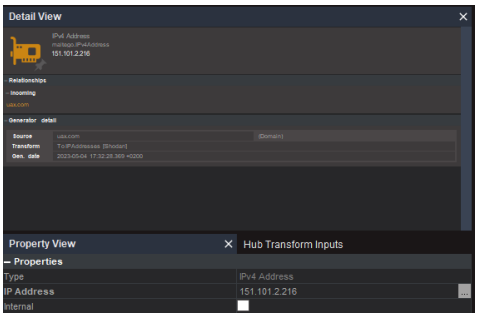
3.3. Análisis de los resultados

3.3.1. Dirección IP

Hemos realizado una investigación en la dirección IP: *151.101.2.216* de la página que estamos analizando y hemos encontrado dos aspectos relevantes.



(a) Dirección IP en el grafo



(b) características de la Dirección IP

En primer lugar, se ha identificado que la dirección IP está siendo gestionada por el proveedor de servicios de red Fastly, el cual es conocido por ofrecer soluciones de entrega de contenido (CDN) y por su alta disponibilidad y escalabilidad en infraestructuras web.

Figura 7: Fastly

Aquí podemos encontrar un nodo donde se ve su relación con esta empresa:

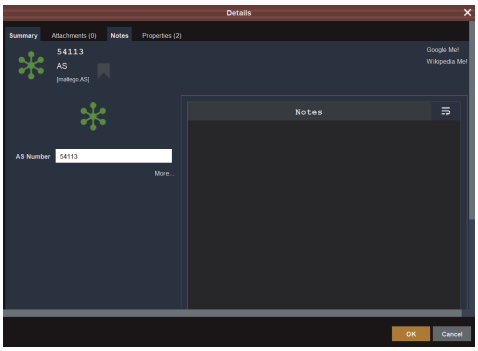


Figura 8: Nodo AS 54113

Encontramos tambien un nodo AS 54113 el cual pertenece a Fastly. Este número 54113 corresponde al Autonomous System (AS) de Fastly Inc., una compañía que ofrece servicios de computación en la

nube, aceleración de contenido y seguridad web. Un AS es un conjunto de redes y sistemas autónomos que comparten una misma política de enrutamiento y están bajo el control de una sola entidad administrativa. Esto nos indica que la dirección IP que estás analizando probablemente está alojada en uno de los centros de datos de Fastly.

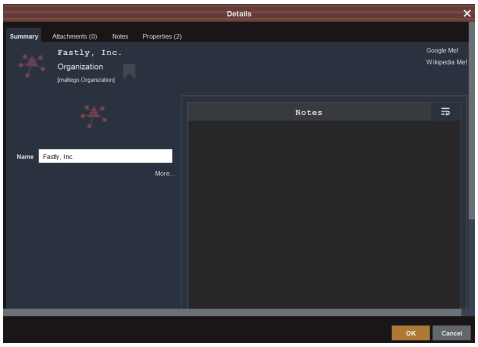


Figura 9: Nodo AS 54113

En segundo lugar, se han encontrado varias búsquedas en el motor de búsqueda Shodan, lo que sugiere que esta dirección IP ha sido indexada en la base de datos de este motor de búsqueda especializado en el análisis de dispositivos conectados a Internet. Estos resultados podrían ser útiles para comprender mejor la infraestructura y los servicios que se ejecutan en esta dirección IP y para identificar posibles vulnerabilidades o riesgos de seguridad asociados.

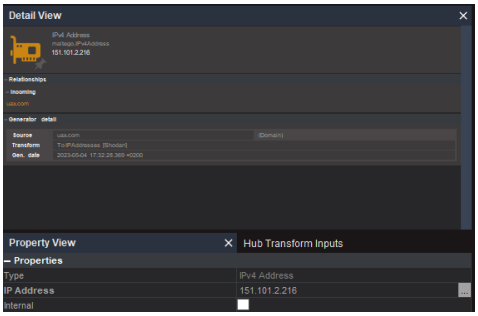


Figura 10: Shodan

3.3.2. Dominios

Los dominios son identificadores únicos utilizados para identificar sitios web en Internet. Un dominio típicamente consta de dos partes: el nombre de dominio (por ejemplo, "google") y la extensión de dominio (por ejemplo, ".com"). Cada dominio se registra en una base de datos central llamada DNS (Sistema de Nombres de Dominio) que asigna una dirección IP única a cada dominio.

Los dominios se utilizan para facilitar el acceso a sitios web y para establecer una presencia en línea. Además de ser utilizados para sitios web, los dominios también se pueden utilizar para correos electrónicos, servidores y otros servicios en línea. Los dominios pueden ser registrados y poseídos por particulares, empresas y organizaciones sin fines de lucro.

Es importante tener en cuenta que los dominios no son lo mismo que las direcciones IP. Mientras que una dirección IP es una serie única de números que identifica una ubicación en línea específica, los

dominios son nombres legibles para los humanos que se utilizan para acceder a dicha ubicación.

En nuestra búsqueda de información sobre UAX, hemos encontrado una serie de dominios relacionados:

- uax.bj.cn
- uax.buzz
- uax.cash
- uax.mobi
- uax.msk.ru
- uax.nl
- uax.nu
- uax.online
- uax.party
- uax.space
- uax.us

Los dominios relacionados con üax"son una serie de nombres de dominio que contienen la misma secuencia de caracteres que el dominio principal üax.com", pero con diferentes extensiones de dominio. Estas extensiones de dominio incluyen ".bj.cn", ".buzz", ".cash", ".mobi", ".msk.ru", ".nl", ".nu", ".online", ".party", ".spacez ".us".

Figura 11: Ejemplo de dominio

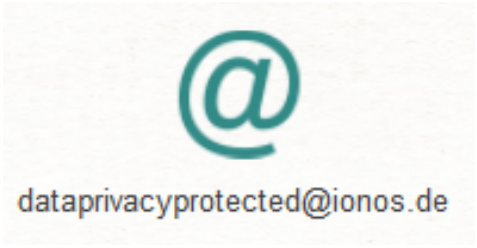
Cada una de estas extensiones de dominio se refiere a una región geográfica o una categoría específica de sitios web. Por ejemplo, ".bj.cn.[es] el dominio de nivel superior de código de país para China, mientras que ".online"se refiere a sitios web que están en línea.

En algunos casos, los propietarios de dominios compran varias extensiones de dominio relacionadas para proteger su marca y asegurarse de que ningún competidor pueda registrar un dominio similar. También puede ser útil registrar varias extensiones de dominio para dirigir tráfico a un sitio web específico y aumentar la visibilidad en línea de una marca.

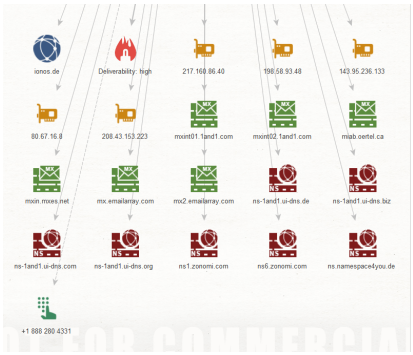
3.3.3. Análisis de correos

En esta sección vamos a analizar varios correos que se encuentran en la pagina de la *UAX*.

Hemos primer lugar, hemos realizado el análisis del correo electrónico enviado desde dataprivacy-protected@ionos.de. subcaption



(a) Dirección de correo en el grafo



(b) Nodos encontrados dentro

donde hemos podido identificar información relevante para la investigación en curso. En particular, hemos encontrado que el correo se envió desde un servidor con dirección IP 74.208.5.23, cuyo dominio asociado es ionos.com. Además, hemos encontrado que el correo cuenta con una etiqueta de IPQS que indica que su entregabilidad es alta.

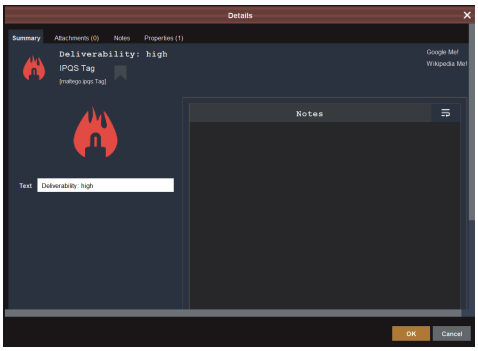
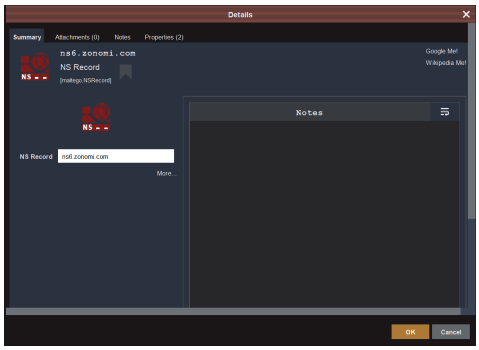
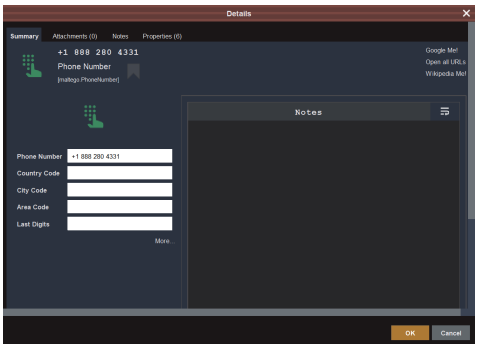


Figura 13: Etiqueta IPQS

Por otro lado, se ha identificado la presencia de un registro NS para el dominio en ns6.zonomi.com, lo que sugiere que este servidor podría estar siendo utilizado para la resolución de DNS. Asimismo, se ha encontrado un número de teléfono asociado a la cuenta de correo en cuestión, el cual tiene un prefijo internacional (+1 888 280 4331).



(a) Dirección de correo en el grafo



(b) Telefono

En definitiva, estos hallazgos pueden ser de gran utilidad para continuar con la investigación y obtener más información sobre la fuente del correo y su posible relación con otros aspectos del caso.

Además, en la página web se han encontrado otros correos electrónicos asociados a la UAX y a sus diferentes departamentos, como por ejemplo:

- abuse@ionos.com
- dataprivacyprotected@ionos.de
- fgabiola@uax.es
- fterrram@uax.es
- gaop@uax.es
- info@openuax.com
- info@uax.es
- inter@uax.es
- last@tiama.com
- lmaicper@uax.es
- mperona@uax.es
- msarddet@uax.es
- oficinadelestudiante@uax.es
- paramejorar@uax.es
- postmaster@uax.es

3.3.4. Company

Durante nuestra investigación, encontramos varias empresas asociadas a la dirección IP de la página que estamos analizando.

Figura 15: Ejemplo de 3 companies

A continuación, vamos a detallar cada una de ellas y su posible relación con el sitio web en cuestión:

- **Admin Organization:** la organización administradora de la dirección IP
- **Registrant City:** la ciudad del registrante de la dirección IP
- **Registrant State/Province:** el estado o provincia del registrante de la dirección IP
- **Registrant Street:** la calle del registrante de la dirección IP
- **UAX.COM:** la empresa propietaria del dominio de la página web
- **VeriSign:** la empresa responsable de los registros de nombres de dominio .com y .net
- **VeriSign Global Registry:** el registro global de nombres de dominio de VeriSign
- **Whois:** un protocolo utilizado para buscar información de contacto en una base de datos de registros de nombres de dominio
- **whois.ionos.com:** el servidor de nombres de dominio utilizado para buscar información de contacto en la base de datos de registros de nombres de dominio.

Es importante destacar que la presencia de estas empresas no implica necesariamente una relación directa con la página web en cuestión, pero su identificación puede ser útil para comprender mejor el contexto y el entorno en el que se encuentra la página web.

3.3.5. Documentos

Realizando el análisis, hemos encontrado varios nodos que corresponden a archivos que estás publicados en la página de la universidad. Los enumeramos:

Figura 16: Ejemplo de documentos

1. Document CONSEJO DE GOBIERNO - UAX: Documento relacionado con el Consejo de Gobierno de la Universidad Alfonso X el Sabio.
2. Document Calendario Académico 2021/22 - GRADOS: Calendario académico para el curso 2021/2022 en los grados de la UAX.

3. Document HONORARIOS ACADÉMICOS DE CICLOS FORMATIVOS DE GRADO SUPERIOR: Información sobre los honorarios académicos para ciclos formativos de grado superior en la UAX.
4. Document HONORARIOS ACADÉMICOS DE CICLOS FORMATIVOS DE GRADO SUPERIOR - UAX: Información sobre los honorarios académicos para ciclos formativos de grado superior en la UAX.
5. Document HONORARIOS ACADÉMICOS DE GRADOS - UAX: Información sobre los honorarios académicos para los grados en la UAX.
6. Document HONORARIOS ACADÉMICOS DE OPENUAX: Información sobre los honorarios académicos para el programa OpenUAX en la UAX.
7. Document HONORARIOS ACADÉMICOS DE POSTGRADOS - UAX: Información sobre los honorarios académicos para los postgrados en la UAX.
8. Document Calendario Académico 2022/23 : Calendario académico para el curso 2022/2023 en la UAX.
9. Document RESIDENCIAS UAX SIÉNTETE COMO EN CASA: Información sobre las residencias de la UAX.
10. Document UAX: Información general sobre la Universidad Alfonso X el Sabio.
11. Document Universidad Alfonso X El Sabio: Universidad Privada en Madrid - UAX: Información general sobre la Universidad Alfonso X el Sabio.
12. Document www.uax.com: Página web de la UAX.

3.3.6. Más información extra

- Nombre del dominio: uax.com
- Fecha de registro del dominio: 1997-05-07
- Propietario actual del dominio: IONOS Inc.
- Tipo de sitio web: Sitio web de una universidad
- Descripción: Sitio web de la Universidad Alfonso X El Sabio (UAX), una universidad privada española ubicada en Madrid, que ofrece cursos de pregrado, posgrado y programas de doctorado en diversas áreas de estudio.
- Dirección física: Universidad Alfonso X El Sabio, Ctra. Villanueva de la Cañada a Brunete, km. 1,800, 28691 Villanueva de la Cañada, Madrid, España
- Números de teléfono:
  - +34 918 109 999
  - +34 919 10 01 70
  - +34 919 10 01 72
  - +34 512 51 07 67
  - +1 877 461 2631
  - +1 610 560 1459
  - 044 2345 5858
  - 91 810 90 00
  - 91 810 92 00
  - 91 810 94 00

4. Conclusión

En resumen, en este trabajo hemos llevado a cabo una prueba de penetración en la página web de la Universidad Alfonso X .ᴱˡ Sabio". Para ello, hemos realizado la fase de reconocimiento o footprinting del sitio, utilizando diversas herramientas como Maltego y Python para obtener información sobre la estructura, arquitectura y posibles vulnerabilidades del mismo.

Gracias a la utilización de Maltego, hemos podido recopilar y visualizar información de múltiples fuentes de datos, lo que nos ha permitido identificar puntos débiles y posibles vulnerabilidades en la seguridad del sitio. Además, hemos utilizado Python para llevar a cabo el proceso de web scraping y obtener información adicional del sitio.

A través de este trabajo, hemos demostrado cómo se puede realizar una prueba de penetración en una página web, y cómo se pueden utilizar diversas herramientas para recopilar información y evaluar la seguridad de un sitio. Este es solo un breve ejemplo de todo lo que se podría hacer con la herramienta de Maltego y otras herramientas de hacking ético, pero esperamos que este trabajo sirva como una introducción a la temática y una motivación para continuar aprendiendo sobre el tema.