

**St. Andrews Georgian University**

Report topic: Analyze Malicious and Benign Websites Dataset   
using Feed-Forward Neural Network

Group members: Giorgi Ghibradze, Levan Gelashvili, Irine Burduli,   
Saba Tchitchinadze, David Sakhvadze

Program: MA in Cybersecurity

Course: Machine Learning for Cybersecurity

Lecturer: Paata Gogishvili

Date and Time: 8 May, 2024, Tbilisi

**Problem description:**

The rapid proliferation of websites on the internet has given rise to various security threats, including malicious websites designed to spread malware, phishing attacks, or engage in other malicious activities. Differentiating between benign and malicious websites has become crucial for protecting users from potential harm. Traditional methods of website classification, such as blacklisting or signature-based detection, often struggle to keep pace with the ever-evolving nature of cyber threats. This is where machine learning techniques, particularly feed-forward neural networks, can play a vital role in detecting and classifying malicious websites.

The ability to accurately classify websites as either malicious or benign is of paramount importance for maintaining a secure online environment. Malicious websites can pose serious threats to users, such as data theft, financial fraud, or system compromise. Failing to detect and block access to these sites can lead to significant consequences, including financial losses, privacy breaches, and reputational damage for individuals and organizations alike. The constantly evolving nature of cyber threats necessitates the development of robust and adaptable detection mechanisms. Traditional approaches relying on static signatures or blacklists often struggle to keep up with the rapid pace of new malicious website creation and obfuscation techniques employed by attackers.

Feed-forward neural networks are a type of artificial neural network capable of learning complex patterns and relationships within data. By leveraging the power of feed-forward neural networks and training them on comprehensive datasets, it becomes possible to develop more effective and proactive defense mechanisms against malicious websites. These models can learn to identify subtle patterns and indicators that may be challenging for humans or rule-based systems to detect, enabling timely and accurate classification of websites as either benign or malicious.

**Dataset:**

“Malicious and Benign Websites” dataset was downloaded from the [www.kaggle.com](http://www.kaggle.com) web-site[[1]](#footnote-1). It consists of following data:

|  |  |
| --- | --- |
| URL: | identification of the URL. |
| URL\_LENGTH: | number of characters in the URL. |
| NUMBER\_SPECIAL\_CHARACTERS: | number of special characters identified in the URL, such as, “/”, “%”, “#”, “&”, “. “, “=” |
| CHARSET: | categorical value and its meaning is the character encoding standard (also called character set). |
| SERVER: | categorical value and its meaning is the operative system of the server got from the packet response. |
| CONTENT\_LENGTH: | content size of the HTTP header. |
| WHOIS\_COUNTRY: | values are the countries we got from the server response (specifically, our script used the API of Whois). |
| WHOIS\_STATEPRO: | categorical variable, its values are the states we got from the server response (specifically, our script used the API of Whois). |
| WHOIS\_REGDATE: | Whois provides the server registration date, so, this variable has date values with format DD/MM/YYY HH:MM. |
| WHOIS\_UPDATED\_DATE: | the last update date from the server analyzed. |
| TCP\_CONVERSATION\_EXCHANGE: | number of TCP packets exchanged between the server and our honeypot client. |
| DIST\_REMOTE\_TCP\_PORT: | number of the ports detected and different to TCP |
| REMOTE\_IPS: | the total number of IPs connected to the honeypot |
| APP\_BYTES: | number of bytes transfered |
| SOURCE\_APP\_PACKETS: | packets sent from the honeypot to the server |
| REMOTE\_APP\_PACKETS: | packets received from the server |
| APP\_PACKETS: | total number of IP packets generated during the communication between the honeypot and the server |
| DNS\_QUERY\_TIMES: | number of DNS packets generated during the communication between the honeypot and the server |
| Type: | categorical variable, its values represent the type of web page analyzed, specifically, 1 is for malicious websites and 0 is for benign websites |

**ML method:**

**Feed-Forward Neural Networks** are a fundamental type of artificial neural network architecture that has been widely studied and applied in various domains. As the name suggests, the data flows in a single direction, from the input layer through one or more hidden layers, and finally to the output layer. This unidirectional flow of information, without any loops or feedback connections, is a defining characteristic of these networks.

* **Input layer** serves as the entry point for raw data from the external environment. This data can take various forms, such as numerical values, images, text, or any other relevant representation. The role of the input layer is to distribute this data to the subsequent hidden layers.
* **Hidden layers**, which can be one or multiple, are responsible for performing computations and transformations on the input data. Each neuron in a hidden layer receives weighted inputs from the previous layer, computes a weighted sum, and then applies a non-linear activation function to the result. This activation function introduces non-linearity into the network, allowing it to model complex, non-linear relationships within the data. Common activation functions used in feed-forward networks include the sigmoid function, rectified linear unit (ReLU), and hyperbolic tangent (tanh).
* **Output** **layer** is the final stage of the network, where the transformed data is mapped to the desired output format. In the case of classification tasks, such as distinguishing between malicious and benign websites, the output layer would typically have one neuron per class, with the highest output value indicating the predicted class. For regression tasks, where the goal is to predict a continuous value, the output layer may have a single neuron.

One of the key strengths of feed-forward neural networks lies in their ability to learn from data through a process called training. During training, the network's weights (the strengths of the connections between neurons) are iteratively adjusted using optimization algorithms, such as gradient descent or its variants. The objective is to minimize the difference between the network's output and the desired output, as specified by the training data. This process is known as backpropagation, where the error is propagated backward through the network, and the weights are updated accordingly.

Feed-forward neural networks can be employed for a variety of tasks, including classification, regression, clustering, and pattern recognition. Their versatility and ability to automatically extract relevant features from the input data make them well-suited for problems where traditional linear models may struggle. It is important to note that while feed-forward neural networks offer powerful pattern recognition capabilities, they also have limitations. They can be computationally expensive, especially for large datasets or deep networks with many layers. Additionally, they may be prone to overfitting, where the network learns the training data too well and fails to generalize to new, unseen data. Techniques such as regularization, dropout, and early stopping can help mitigate this issue.

In the context of analyzing malicious and benign websites, feed-forward neural networks can be trained on a dataset containing features extracted from both types of websites. These features could include various characteristics such as URL patterns, website content, network traffic patterns, or any other relevant indicators. By learning from this data, the neural network can develop an understanding of the patterns and characteristics that distinguish malicious websites from benign ones, enabling accurate classification during the operational phase.

**Python code:**

This code does:

1. Import and review data set;
2. Remove individual and redundant features;
3. Encode categorical data;
4. Handle missing values - change "none" with data.median;
5. Normalize data;
6. Split dataset in training and test datasets;
7. Define model architecture;
8. Compile the model;
9. Train the model on training dataset;
10. Use model to validate test dataset;
11. Evaluate model;
12. Analyze test dataset using ROC/AUC, confusion matrix, predicting probability for test set, classification report;
13. Build plots illustrating analysis.

import pandas as pd

import seaborn as sns

import tensorflow as tf

import matplotlib.pyplot as plt

from sklearn.metrics import classification\_report, confusion\_matrix, roc\_auc\_score, roc\_curve

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder, StandardScaler

from tensorflow.keras.layers import Dense

from tensorflow.keras.models import Sequential

# Step 1: Import and review dataset

data = pd.read\_csv("dataset.csv")

print(data.head())

# Step 2: Remove individual and redundant features

features\_to\_remove = ['URL', 'WHOIS\_STATEPRO', 'WHOIS\_REGDATE', 'WHOIS\_UPDATED\_DATE']

data.drop(features\_to\_remove, axis=1, inplace=True)

# Step 3: Encode categorical data

data = pd.get\_dummies(data, columns=['CHARSET', 'SERVER', 'WHOIS\_COUNTRY'])

# Step 4: Handle missing values

data.fillna(data.median(), inplace=True)

# Step 5: Normalize data

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

data[data.columns[:-1]] = scaler.fit\_transform(data[data.columns[:-1]])

# Step 6: Split dataset into training and test datasets

from sklearn.model\_selection import train\_test\_split

X = data.drop('Type', axis=1)

y = data['Type']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Step 7: Define model architecture

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

model = Sequential([

Dense(32, activation='relu', input\_shape=(X\_train.shape[1],)),

Dense(16, activation='relu'),

Dense(1, activation='sigmoid')

])

# Step 8: Compile the model

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Step 9: Train the model on training dataset

history = model.fit(X\_train, y\_train, epochs=10, batch\_size=32, validation\_split=0.2)

# Step 10: Use model to validate test dataset

y\_pred = model.predict(X\_test)

# Step 11: Evaluate model

loss, accuracy = model.evaluate(X\_test, y\_test)

print(f'Test Loss: {loss}, Test Accuracy: {accuracy}')

# Step 12: Analyze test dataset using ROC/AUC, confusion matrix, predicting probability for test set, classification report

# ROC/AUC

roc\_auc = roc\_auc\_score(y\_test, y\_pred)

print(f'ROC AUC: {roc\_auc}')

# Confusion matrix

conf\_matrix = confusion\_matrix(y\_test, y\_pred.round())

print('Confusion Matrix:')

print(conf\_matrix)

# Classification report

class\_report = classification\_report(y\_test, y\_pred.round())

print('Classification Report:')

print(class\_report)

# Step 13: Build plots illustrating analysis

# Plot training and validation loss

plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.title('Training and Validation Loss')

plt.legend()

plt.show()

# Plot ROC curve

fpr, tpr, \_ = roc\_curve(y\_test, y\_pred)

plt.plot(fpr, tpr)

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve')

plt.show()

**Analysis:**

We have analyzed Malicious and Benign Websites dataset (downloaded from the [www.kaggle.com](http://www.kaggle.com) web-site) using the Feed-Forward Neural Network.

**Overall, this model has shown a high effectiveness, giving these scores:**

**Test Loss**: 0.20925773680210114  
**Test Accuracy**: 0.9299719929695129   
**ROC AUC**: 0.9080481865948994

The initial dataset contained 21 columns (features) and 1782 rows (records).

Our Label was “Type”, that according to dataset description is “categorical variable, its values represent the type of web page analyzed, specifically, 1 is for malicious websites and 0 is for benign websites”.



This was the sequence of dataset analysis using feed-forward neural network:

1. Import and review data set;

2. Remove individual and redundant features;

3. Encode categorical data;

4. Handle missing values - change "none" with data.median;

5. Normalize data;

6. Split dataset in training and test datasets;

7. Define model architecture;

8. Compile the model;

9. Train the model on training dataset;

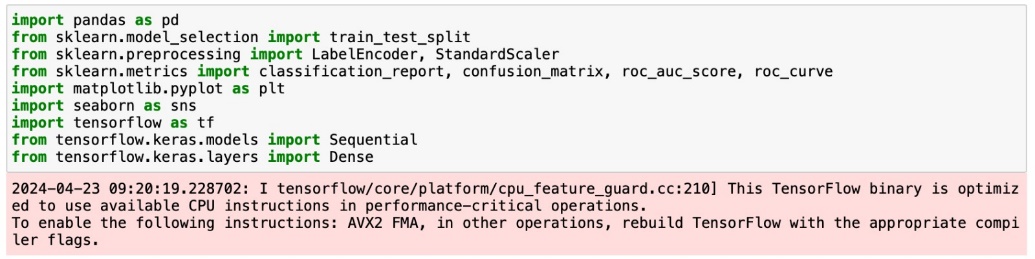
10. Use model to validate test dataset;

11. Evaluate model;

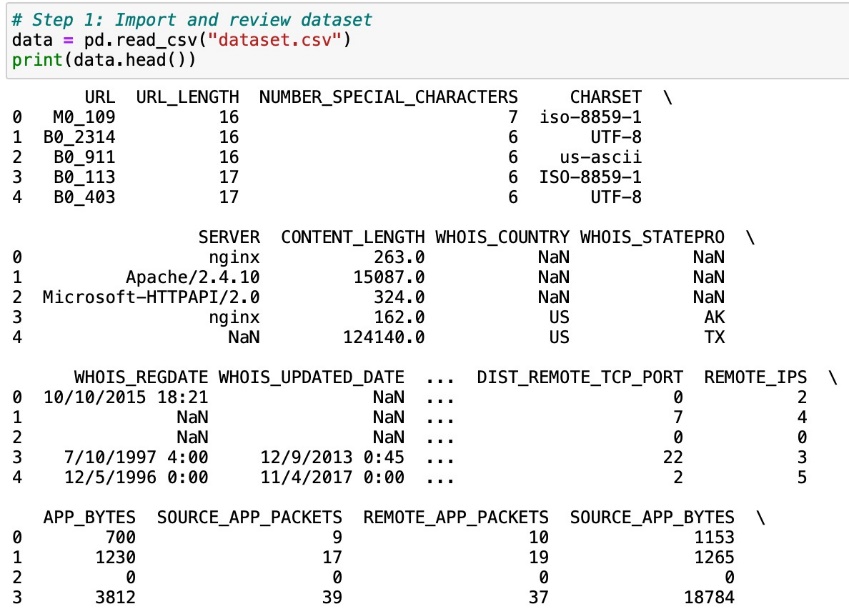
12. Analyze test dataset using ROC/AUC, confusion matrix, predicting probability for test set, classification report;

13.Build plots illustrating analysis.

When importing python libraries, we received a warning:



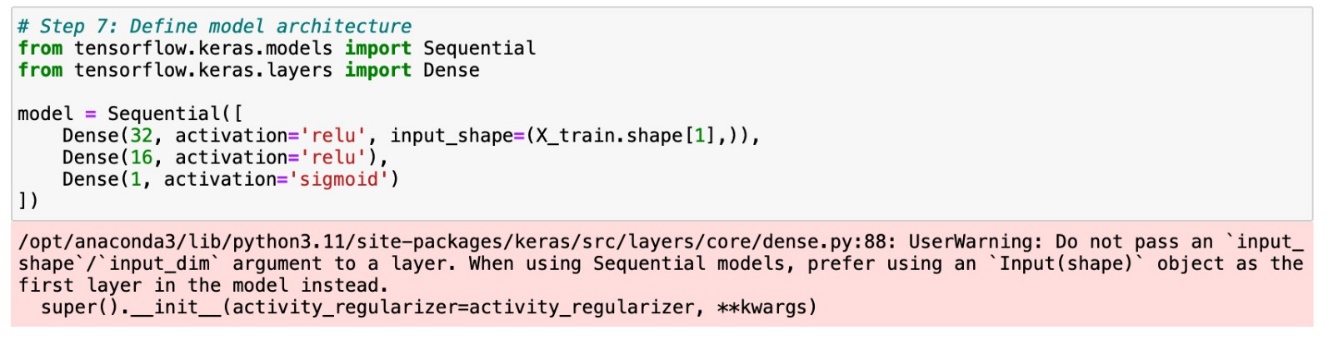
On the Step #1, imported and reviewed data set.  
 (picture is cropped, not showing all the features).



Steps #2-6 completed correctly. On these steps we’ve:

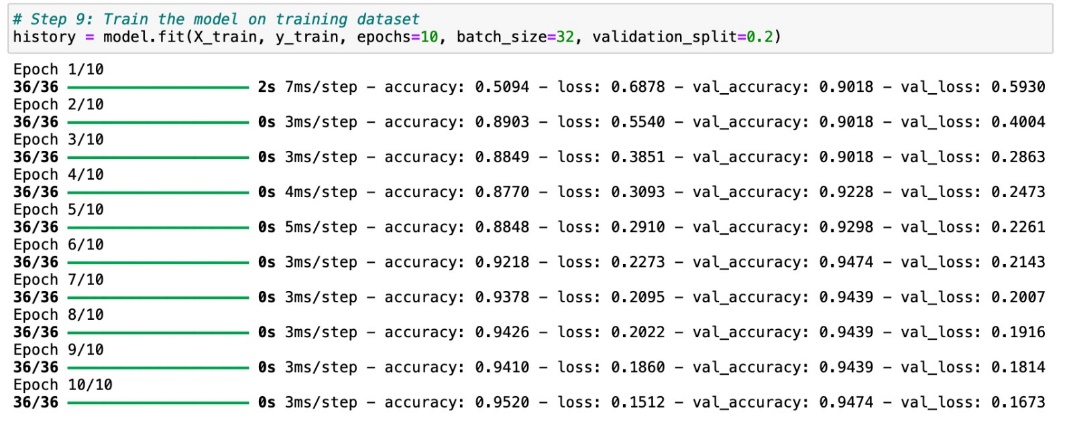
* Removed individual and redundant features;
* Encoded categorical data;
* Handled missing values;
* Normalize data;
* Splitted dataset in training and test datasets;

On Step #7, when defining model, we received a warning:

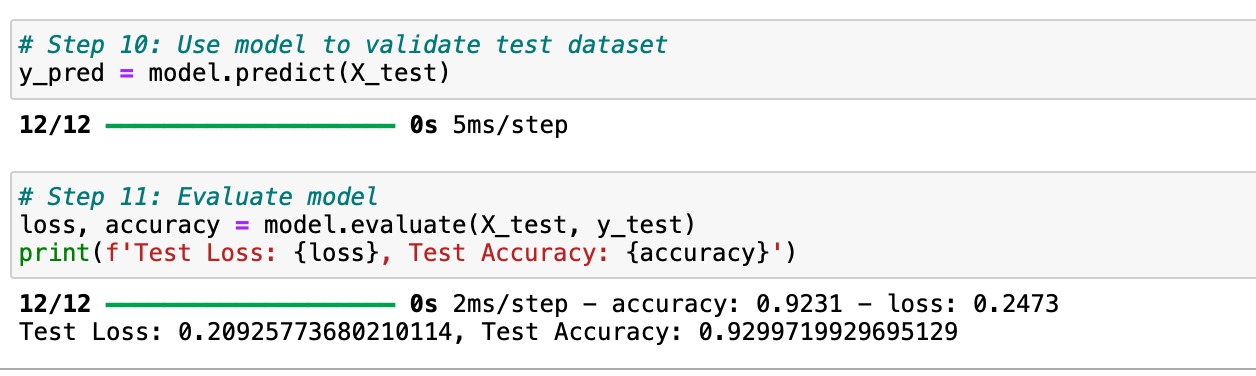


Step #8, model compiling, completed correctly.

Step #9, displayed process of model fitting on training dataset.

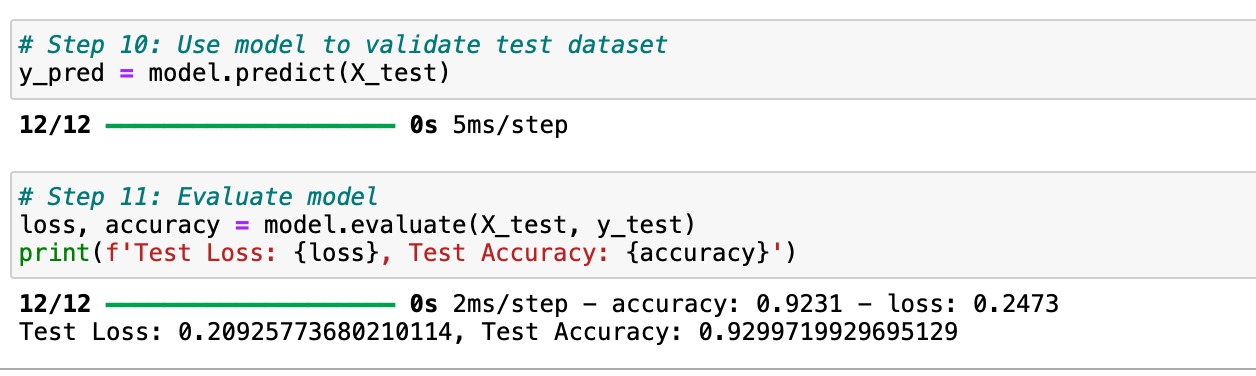


Step #10 displayed model validation of test dataset.



Step #11 displayed model evaluation results.

Test Loss: 0.20925773680210114, Test Accuracy: 0.9299719929695129



Step #12 displayed results of test dataset analysis using ROC/AUC, confusion matrix, predicting probability for test set, classification report.

ROC AUC: 0.9080481865948994

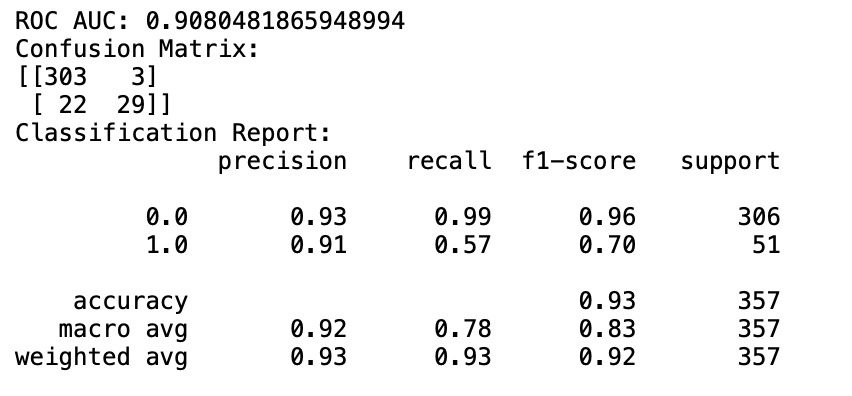
Confusion Matrix:

[[303 3]

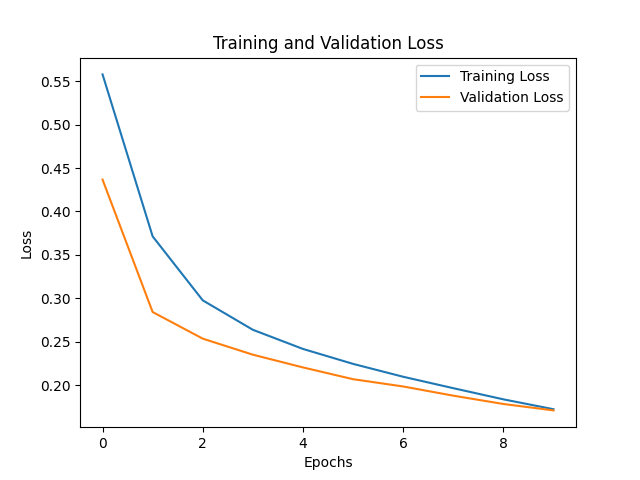
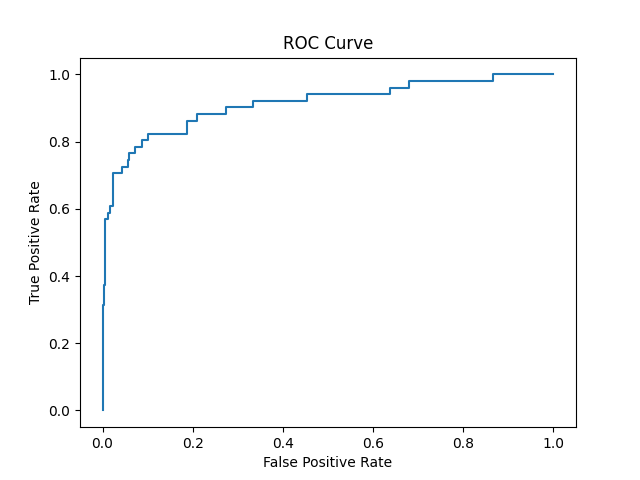
[ 22 29]]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Classification Report: | | | | |
|  | precision | recall | f1-score | support |
| 0.0 | 0.93 | 0.99 | 0.96 | 306 |
| 1.0 | 0.91 | 0.57 | 0.70 | 51 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| accuracy |  |  | 0.93 | 357 |
| macro avg | 0.92 | 0.78 | 0.83 | 357 |
| weighted avg | 0.93 | 0.93 | 0.92 | 357 |



Step#13 displayed diagrams illustarting analysis

****

**References:**

<https://www.kaggle.com/datasets/xwolf12/malicious-and-benign-websites>

[https://claude.ai/chat](https://claude.ai/chat/8df7a7db-a4df-42e3-98d7-8d665219194d)

[https://chat.openai.com/](https://chat.openai.com/c/69576eda-b676-48b6-b0c6-f817b2129114)

1. <https://www.kaggle.com/datasets/xwolf12/malicious-and-benign-websites> [↑](#footnote-ref-1)