Adversarial Attacks against Phishing Website Detectors

Kamal Acharya

April 26, 2026

1 Introduction

Phishing website detection is a vital aspect of cybersecurity, aimed at identifying and mitigating fraudulent websites that trick users into revealing sensitive information. The objective is to continuously monitor and detect phishing websites in real-time. This technology is essential in various domains, such as web security, email security, and online banking.

There are several approaches to phishing detection, including heuristic-based methods, blacklist-based methods, and machine learning-based detectors. Among the machine learning approaches, neural network-based models are prominent for their ability to learn patterns that differentiate legitimate websites from phishing ones. However, neural networks are vulnerable to adversarial attacks—subtle manipulations of input data that can deceive the model. This tutorial will guide you through understanding and implementing adversarial attacks against machine learning-based phishing website detectors (ML-PWDs). We will focus on a recent study, "SpacePhish: The Evasion-space of Adversarial Attacks against Phishing Website Detectors using Machine Learning,"[2] which provides a comprehensive framework for these attacks.

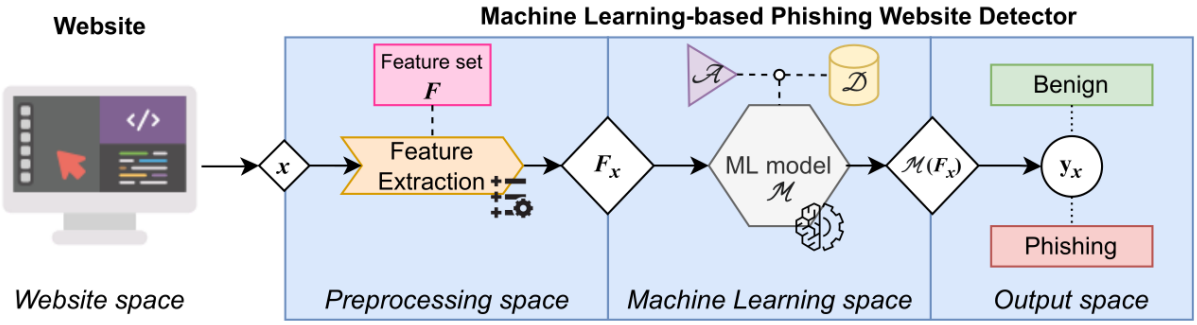

Figure 1: Architecture of a ML-PWD

2 Understanding the Security Problem

Phishing website detection systems are essential in protecting users from malicious sites that deceive them into revealing sensitive information, such as passwords and credit card numbers. These systems typically employ machine learning models to differentiate between legitimate and phishing websites based on features extracted from the URLs and the website content. Despite their effectiveness, these models are inherently vulnerable to adversarial attacks. Adversarial attacks exploit the susceptibility of machine learning algorithms by introducing slight, often imperceptible perturbations to the input data, which can mislead the model into making incorrect classifications. For instance, a phishing URL might be slightly modified in a way that preserves its malicious intent but causes the detector to classify it as benign. This vulnerability poses a significant threat to the reliability of phishing website detectors, as attackers can craft inputs that evade detection, thereby undermining the security provided by these systems. Addressing this issue requires a deep understanding of the attack vectors and the development of robust defenses to mitigate the impact of such adversarial examples.

3 Overview of Phishing Website Detectors

The architecture of a ML-PWD consists of four main spaces: website space, preprocessing space, and machine learning space, ultimately leading to the output space as given in the Figure 1.

- **Website Space:** This is the initial stage where the input x, representing a website,

is considered for analysis. The website's content, structure, and other relevant characteristics are prepared for further processing.

- **Preprocessing Space:** Here, feature extraction is performed on the input x. The feature set F encompasses various attributes of the website, such as URL length, presence of URL shorteners, HTML object ratios, and hidden inputs. These features F_x are extracted and transformed into a structured format suitable for machine learning analysis. This space ensures that the raw website data is converted into meaningful features that capture the essence of phishing indicators.

- **Machine Learning Space:** In this stage, the processed feature set F_x is fed into the machine learning model $\mathcal{M}$. The ML model is trained on a labeled dataset D and employs an algorithm A to learn patterns and make predictions. The model analyzes the features to determine the likelihood of the website being phishing or benign.

- **Output Space:** The output of the machine learning model $\mathcal{M}(F_x)$ is a prediction y_x which classifies the website as either benign or phishing. The decision is based on the analysis performed in the machine learning space, providing a clear and actionable outcome for users or automated systems to act upon.

The ML model $\mathcal{M}$ predicts the ground truth of F_x as y_x, expressed as $\mathcal{M}(F_x) = y_x$. Thus, the workflow of our ML-PWD can be summarized by the following sequence:

$$x \rightarrow F_x \rightarrow \mathcal{M}(F_x) = y_x \tag{1}$$

If x is a phishing (benign) webpage and y_x is also phishing (benign), then the result is a true positive (true negative). Conversely, if the classification is incorrect, it results in either a false positive or a false negative. We assume that $\mathcal{M}$ has been properly trained to ensure high performance, yielding a high true positive rate (tpr) and maintaining a low false positive rate (fpr) provided no adversarial attacks are present.

4 Adversarial Attack on Phishing Website Detectors

Adversarial attacks exploit perturbations, denoted as ϵ, that manipulate an ML model $\mathcal{M}$ to produce an output advantageous to the attacker. In this context, $\mathcal{M}$ is a binary classifier analyzing F_x. Therefore, an adversarial attack can be expressed as finding ϵ such that $\mathcal{M}(F_x + \epsilon) = y_{x\epsilon} \neq y_x$. In other words, the goal is to find a perturbation ϵ that causes the ML model $\mathcal{M}$ (assumed to be accurate) to misclassify a given sample x (i.e., $y_{x\epsilon} \neq y_x$). In the case of evasion attacks, this misclassification results in a phishing website being classified as benign. The perturbation ϵ must (i) preserve the ground truth (i.e., $y_{x\epsilon}$ should match y_x) and (ii) maintain the phishing characteristics of the webpage. However, the impact of ϵ on $y_{x\epsilon}$ depends on where it is introduced in the workflow described by Equation 1.

4.1 Evasion-Space

The evasion-space is divided into four 'spaces', with perturbations in the first three spaces being relevant for adversarial attacks. Perturbations in the output-space do not qualify as adversarial ML attacks since they are unrelated to the ML model $\mathcal{M}$.

- Website-space Perturbations (WsP): The detection workflow starts in the website-space, where the website x is generated. This space is accessible to attackers who can modify the URL or the website's representation. A perturbation ϵ in this space (i.e., WsP) results in an adversarial sample $x' = x + \epsilon$, potentially affecting all subsequent operations of the ML-PWD. However, whether ϵ influences the detection depends on the ML-PWD's implementation.

- Preprocessing-space Perturbations (PsP): After x is acquired by the ML-PWD, it is transformed into F_x. An attacker with access to the preprocessing-space can introduce a PsP ϵ that modifies the feature representation, resulting in $F'_x = F_x + \epsilon$. For example, an attacker could change the URL length feature. While PsP are guaranteed to be noticed by the ML-PWD, they may not necessarily influence the model's predictions.

- ML-space Perturbations (MsP): Once the preprocessing is complete, F_x enters the machine learning-space for analysis by $\mathcal{M}$. An attacker with access to this space can introduce an MsP ϵ, modifying F_x just before it reaches $\mathcal{M}$, resulting in $F'_x = F_x + \epsilon$. MsP are the most powerful perturbations as they bypass all integrity checks, potentially leading to significant misclassifications.

Any perturbation ϵ should ultimately affect the feature representation F_x of a sample x. The key is determining where the perturbation is introduced, which can occur in three spaces: WsP, PsP, and MsP. The corresponding expressions are:

$$\text{find } \epsilon \text{ s.t.} \begin{cases} \overline{x} = x + \epsilon \Rightarrow x \rightarrow \overline{x} \rightarrow F_{\overline{x}} \rightarrow \mathcal{M}(F_{\overline{x}}) = y_x^\epsilon \neq y_x, & \text{WsP;} \\ \overline{F_x} = F_x + \epsilon \Rightarrow x \rightarrow F_x \rightarrow \overline{F_x} \rightarrow \mathcal{M}(\overline{F_x}) = y_x^\epsilon \neq y_x, & \text{PsP;} \\ \overline{F_x} = F_x + \epsilon \Rightarrow x \rightarrow F_x \rightarrow F_x \rightarrow \overline{F_x} \rightarrow \mathcal{M}(\overline{F_x}) = y_x^\epsilon \neq y_x, & \text{MsP.} \end{cases} \tag{2}$$

While some MsP may not be physically realizable in the website-space, it is unfair to consider all MsP or PsP as non-physical. The cost of implementing these perturbations increases from WsP to PsP to MsP, as deeper access to the ML-PWD is required for MsP.

5 Experimental Setup

We will evaluate the robustness of 18 Machine Learning-based Password Detectors (ML-PWD) against 12 evasion attacks. These attacks are designed based on our threat model and executed in various evasion spaces.

5.1 Testbed

We evaluate 18 ML-PWD, which differ based on the source dataset (2 types), the ML algorithm (3 types), and the feature set (3 types) used to develop each model.

Table 1: MStatistics and state-of-the-art of our datasets.

Dataset	Benign	Phish	fpr	tpr
δphish[3]	5511	1012	0.01	0.98
δphish[3]	2000	2000	0.08	0.99

5.1.1 Source Datasets

We rely on two datasets for ML-PWD: $\delta Phish$[3] and $Zenodo$[12]. This choice is driven by three main reasons:

- Both datasets contain raw information for each sample (specifically, the URL and HTML). This is crucial because most of our attacks exploit WsP, which requires modifying the raw webpage before feature extraction.
- Both datasets have been validated by prior research[3][12].
- They ensure experimental reproducibility. Collecting ad-hoc data from public feeds (e.g., AlexaTop/PhishTank) hinders fair future comparisons since phishing webpages are quickly taken down, and it becomes impossible to retrieve complete information on webpages 'blocklisted' years ago.

We provide an overview of our datasets in Table 1, showing the number of samples (benign and phish) and the performance (tpr and fpr) achieved by their creators (without evasion). It is noteworthy that the original *Zenodo* dataset contains 100k phishing and nearly 4M benign webpages. To make our evaluation manageable, we randomly sample 4000 webpages from Zenodo, evenly split between benign and phishing. This approach allows us to analyze the ML-PWD's response with different balancing: while *Zenodo* is perfectly balanced, δphish has significantly more benign samples.

5.1.2 ML Algorithms

We will evaluate ML-PWD using both shallow and deep learning algorithms[1]. Our selection aims to provide a comprehensive assessment of ML-PWD based on representative ML methods. We consider:

- Logistic Regression (LR): As one of the simplest ML algorithms, LR is included because it was believed to be used by the ML-PWD embedded in Google Chrome[7].
- Random Forests (RF): This ensemble technique often outperforms other contenders for ML-PWD[11].
- Convolutional Neural Network (CNN): We include this well-known deep learning technique[6] due to its proven efficacy in ML-PWD[13].

  All these algorithms support binary classification, making them suitable for our ML-PWD evaluation.

5.1.3 Feature Sets

We consider ML-PWD that utilize three feature sets (F), all similar to the one described in our use-case. Specifically, our ML-PWD analyze one of the following:

- URL-only (F^u): The first 35 features listed in Table 2.
- Representation-only (F^r): The last 22 features listed in Table 2.
- Combined (F^c): All features in Table 2.

Analyzing more information (i.e., larger feature sets like F^c) generally leads to better detection performance, as demonstrated in studies like [3]. However, in some scenarios, using larger feature sets may not be feasible. Our feature sets are not only popular in research [4][5][8][10] but also in practice. For instance, several leading security companies organize MLSEC, an ML evasion competition, annually. In 2021 and 2022, MLSEC included challenges to evade ML-PWD that specifically analyzed the HTML representation of a webpage i.e., our F^r.

5.2 Technical Implementation

We now describe how we integrated all the previously mentioned elements to develop our "baseline" ML-PWD. A schematic of our workflow is presented in Figure 2. Each source dataset ($Zenodo$ and $\delta Phish$) represents a distinct setting, which we use to extract the

Table 2: Features F of the considered ML-PWD.

#	Feature Name	#	Feature Name	#	Feature Name
1	URL_length	20	URL_shrtWordPath	39	HTML_commPage
2	URL_hasIPaddr	21	URL_lngWordURL	40	HTML_commPageFoot
3	URL_redirect	22	URL_DNS	41	HTML_SFH
4	URL_short	23	URL_domAge	42	HTML_popUp
5	URL_subdomains	24	URL_abnormal	43	HTML_rightClick
6	URL_atSymbol	25	URL_ports	44	HTML_domCopyright
7	URL_fakeHTTPS	26	URL_SSL	45	HTML_nullLnkWeb
8	URL_dash	27	URL_statisticRe	46	HTML_nullLnkFooter
9	URL_dataURI	28	URL_pageRank	47	HTML_brokenLnk
10	URL_commonTerms	29	URL_regLen	48	HTML_loginForm
11	URL_numerical	30	URL_checkGI	49	HTML_hiddenDiv
12	URL_pathExtend	31	URL_avgWordPath	50	HTML_hiddenButton
13	URL_punyCode	32	URL_avgWordHost	51	HTML_hiddenInput
14	URL_sensitiveWrd	33	URL_avgWordURL	52	HTML_URLBrand
15	URL_TLDinPath	34	URL_lngWordPath	53	HTML_iframe
16	URL_TLDinSub	35	URL_lngWordHost	54	HTML_favicon
17	URL_totalWords	36	HTML_freqDom	55	HTML_statBar
18	URL_shrtWordURL	37	HTML_objectRatio	56	HTML_css
19	URL_shrtWordHost	38	HTML_metaScripts	57	HTML_anchors

corresponding training and inference partitions for our ML-PWD. These ML-PWD are based on one of three ML algorithms, encompassing either shallow (LR and RF) or deep learning (CNN) classifiers. Each classifier is available in three variants, depending on the features analyzed (F^u, F^r, or F^c), resulting in a total of 9 "baseline" ML-PWD per source dataset. We ensure that these 9 ML-PWD optimize their performance, aiming for high true positive rate (tpr) and low false positive rate (fpr), particularly for F^c.

5.2.1 Feature Extractor

A crucial component of our evaluation is the feature extractor. For this, we follow the established guidelines provided in[8][9], which are still widely used in recent literature[5]. These guidelines are based on analyzing various elements of a webpage (e.g., the length of its URL) and using threshold-based mechanisms to classify these elements as 'benign' or 'phishing' (e.g., a short URL is likely benign, whereas a long one is likely phishing). Each feature can take a value within the range [-1, 1], where -1 indicates 'benign' and 1 indicates 'phishing'. Our extractor generates all the features listed in Table 2. We explain

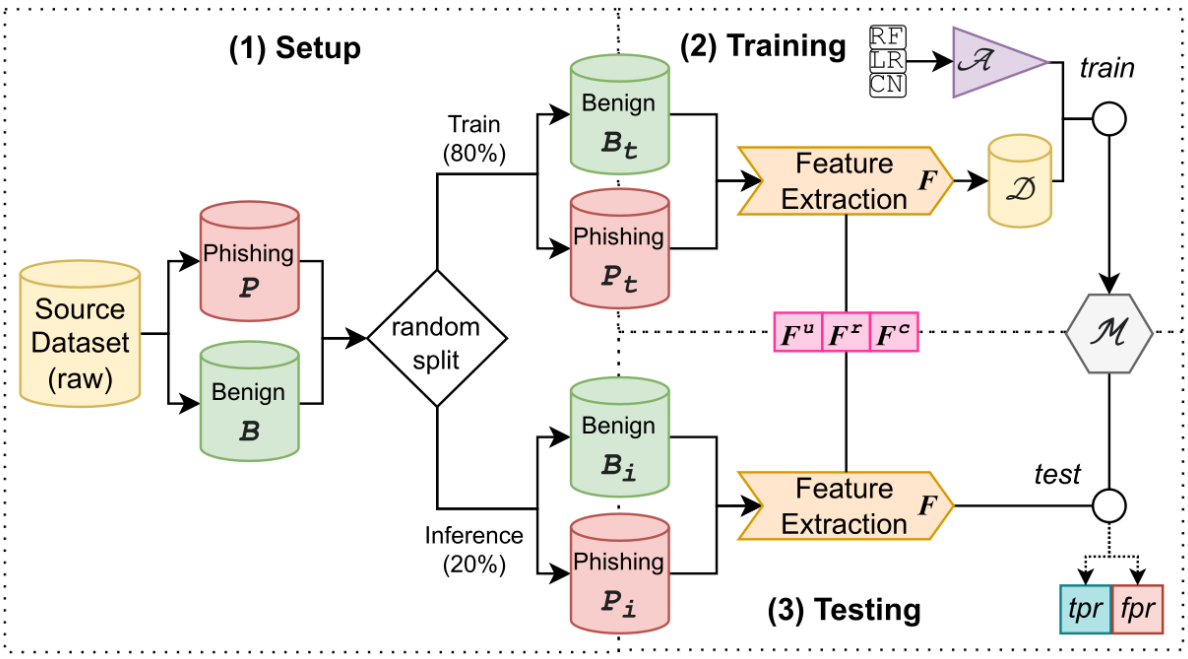

Figure 2: Model of the research

some of them in detail below.

- (#1) URL_length: We measure the number of characters in the entire URL. Strings shorter than 53 characters are assigned a value of -1 (likely benign), while longer strings are assigned a value of +1 (likely phishing).

- (#4) URL_short: If the URL begins with keywords associated with popular URL shortening services (e.g., bit.ly, goo.gl, tinyurl, ad.fly), this feature is set to +1. Otherwise, it is set to -1.

- (#28) URL_pageRank: Using the OpenPageRank API, we query the URL domain. The response gives page ranks from 0 to 10, which we normalize between -1 (if the rank is 10) and +1 (if the rank is 0).

- (#37) HTML_objectRatio: We capture all objects embedded in the webpage and compute the ratio of internal to external objects. An internal object either starts with ../ or shares the same root as the website's URL. If the ratio is less than 0.15, this feature is set to -1 (likely benign); otherwise, it is set to +1 (likely phishing).

- (#38) HTML_metaScripts: Similar to #37, but for scripts, links, and meta tags. If

the ratio exceeds 0.61, the feature is set to +1 (likely phishing); if it is below 0.52, the feature is set to -1 (likely benign); otherwise, it is set to 0.

- (#45) HTML_nullLnkWeb: We count the number of useless links that point to the exact same page (e.g., href=#). The count is normalized between +1 (high number of useless links) and -1 (no useless links).

- (#51) HTML_hiddenInput: We check for hidden input tags in the webpage. If such tags are present, the feature is set to +1 (likely phishing); if not, it is set to -1 (likely benign).

- (#52) HTML_URLBrand: We check if the webpage title includes the brand name in the URL. If it does, the feature is set to -1 (benign); otherwise, it is set to +1 (phishing).

5.2.2 Development of the ML-PWD

The development of the Machine Learning Phishing Website Detector (ML-PWD) consists of three main phases, illustrated in Figure 2:

1. Setup:

   Select a source dataset (e.g., *Zenodo* or *δphish*) and partition its samples into benign (B) and phishing (P). Perform a random split on each partition to avoid bias, using an 80:20 ratio. The 80% of samples in both B and P (denoted as B_t and P_t, respectively) are used to train the ML model, while the remaining 20% (B_i and P_i) are reserved for assessing the model's inference performance. Additionally, P_i will be used to craft adversarial samples.

2. Training:

   Transform the raw source data into its feature representation before obtaining the training dataset (D). This involves developing a feature extractor based on a given feature set (F^u, F^r, or F^c). Preprocess B_t and P_t to obtain the actual training data (D). Apply a chosen ML algorithm (A) such as Random Forest (RF), Logistic

Regression (LR), or Convolutional Network (CN) to D. The resulting ML model ($\mathcal{M}$), a binary classifier, is fine-tuned via grid search and serves as the detection component of the ML-PWD.

3. Testing:

   Measure the performance of $\mathcal{M}$, ensuring a high detection rate and a low false positive rate to avoid blocking legitimate websites due to false alarms. Preprocess the inference partitions B_i and P_i using the appropriate feature set (F) and measure the false positive rate (fpr) and true positive rate (tpr) in the absence of adversarial attacks.

6 Steps to Run the Model

We need to first install some prerequisites.

6.1 Prerequisites

- Download and install VS Code https://code.visualstudio.com/download
- Download and install git https://git-scm.com/downloads
- Download and install python https://www.python.org/downloads/

6.2 Clone the Repository

1. Open VS Code
2. Open Terminal: Go to View > Terminal or press "Ctlr +" to open the integrated terminal.
3. Clone the Repository by running the code:

```
git clone https://github.com/lotussavy/CyberSecurityTraining2024.git
```

4. Navigate to the Directory:

```
cd CyberSecurityTraining2024
```

6.2.1 Structure of Repository

This repository includes two main folders:

- **ml_folder:** It contains the source-code of our main experiments and has 3 files:
  - extractor.py: this python script analyzes a sample (a phishing webpage) and extracts its feature representation.
  - feature_extraction.ipynb: is a (small) notebook showcasing the application of extractor.py on a single sample.
  - PA_PSP.ipynb: is a notebook that applies the perturbations related to the preprocessing attacks considered in our paper.
- **preprocessing_folder:** It contains the code of our feature extractor and some attacks. This folder contains 4 files, which refer to the experiments performed on the "DeltaPhish" dataset.
  - ML.py, containing some custom-defined functions for developing our ML models and printing their results
  - RF/CN/LR_experiments.ipynb, which are notebooks containing the experiments for each of the 3 main ML algorithms (RF=random forest, LR=logistic regression, CN=convolutional neural network).

In the root folder of this repository, we have also provided a "requirements.txt" file, specifying which Python libraries will be used to carry out all our experiments. Moreover, we also provided a document ("get_data.md") explaining how to retrieve the data for our experiments. This artifact entirely runs on CPU. We also have the original paper "SpacePhish.pdf" on which we are working on.

6.3 Set Up the Python Environment

1. Create a Virtual Environment named venvPWD:

```
python -m venv venvPWD
```

2. Activate it: On Windows:

```
.\venvPWD\Scripts\activate
```

On macOS/Linux:

```
source venvPWD/bin/activate
```

After activation you will see name (venvPWD) in your terminal

3. Install Dependencies:

```
pip install -r requirements.txt
```

6.4 Download the Dataset

Since the dataset is too large to put in the github, we uploaded it on google drive. Here is the link: https://drive.google.com/file/d/1RV0soZrrNZjH2EOqtTnHygvohow_C_uQ/view?usp=sharing

6.4.1 Preliminary Information

We will be using two datasets: δphish[3] and δphish[3] (both of which are publicly available), containing "raw data" of webpages (benign or phishing). For transparency, we include in this repository all such "raw data", which will be deleted (to avoid potential copyright violations). We will, however, maintain the preprocessed version of each webpage.

Our work entails "adversarial attacks against machine learning", whose basic principle is to (i) take a sample, (ii) manipulate such sample in some way, and (iii) assess whether the "adversarial sample" evades a given ML model or not. Specifically in our case, we consider a total of 12 adversarial attacks, meaning that we artificially create 12 "adversarial variants" of each "original sample" (i.e., a phishing webpage). Some of these variants are created "at runtime", whereas the others are created "in advance" (we did this by manually manipulating each raw sample).

6.4.2 Structure

The folder is organized depending on the dataset(deltaphish or zenodo), the format (raw or preprocessed). Let us explain both of these:

- **raw:** This folder contains the "original" data as well as the adversarial variants of each sample.
  - **normal:** This folder contains information on the "original" webpages. It contains a JSON file with the URLs of each sample; and an "HTML" folder containing the raw HTML of each sample
  - **wa:** This folder refers to the "cheap" attacks considered in our paper. It contains files including the HTML of phishing webpages after applying the "cheap" HTML manipulation.
  - **wa+:** This folder refers to (a subset of) the "advanced" attacks of our paper. It contains files including the HTML of phishing webpages after applying the "advanced" HTML manipulations.
- **Preprocessed:** This folder includes data describing the "preprocessed" format of each sample in the "raw" folder after the application of our custom-built feature extractor.
  - **normal:** This folder contains a single JSON file describing the feature representation of each "normal" sample (benign and phishing)
  - **wa and wa+:** These folders contain three subfolders ("u, r, c") each referring to a specific variant of our wa/wa+ attacks. Each subfolder has a single JSON file, which contains the feature representation of each (phishing) sample after applying adversarial manipulation.
  - **phish_sub_test_x_100.pkl :** This is a "pickle" file including the 100 samples used as basis for our our adversarial attacks in the preprocessing space.

6.5 Run the Application

1. **Get the data, and install requirements.** This part is covered in the above section. Important: the data_folder should be placed in the root directory!

2. **Test the feature extractor.** Simply run the preprocessing_folder/feature_extraction.ipynb notebook once. It should prove that the feature extractor "works".

3. **Create the adversarial samples.** Simply run the preprocessing_folder/PA_PSP.ipynb once.

4. **Test the attacks.** Consider any of the three notebooks in ml_folder and run all of its cells. The LR and RF do not take long to train, whereas the CN can take several minutes. All of the code can be run in the CPU based systems if GPU is not available.

References

[1] Giovanni Apruzzese, Michele Colajanni, Luca Ferretti, and Mirco Marchetti. Addressing adversarial attacks against security systems based on machine learning. In *2019 11th international conference on cyber conflict (CyCon)*, volume 900, pages 1–18. IEEE, 2019.

[2] Giovanni Apruzzese, Mauro Conti, and Ying Yuan. Spacephish: the evasion-space of adversarial attacks against phishing website detectors using machine learning. In *Proceedings of the 38th Annual Computer Security Applications Conference*, pages 171–185, 2022.

[3] Igino Corona, Battista Biggio, Matteo Contini, Luca Piras, Roberto Corda, Mauro Mereu, Guido Mureddu, Davide Ariu, and Fabio Roli. Deltaphish: Detecting phishing webpages in compromised websites. In *Computer Security–ESORICS 2017: 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part I 22*, pages 370–388. Springer, 2017.

[4] Abdelhakim Hannousse and Salima Yahiouche. Towards benchmark datasets for machine learning based website phishing detection: An experimental study. *Engineering Applications of Artificial Intelligence*, 104:104347, 2021.

[5] Ankit Kumar Jain and Brij B Gupta. Towards detection of phishing websites on client-side using machine learning based approach. *Telecommunication Systems*, 68:687–700, 2018.

[6] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[7] Bin Liang, Miaoqiang Su, Wei You, Wenchang Shi, and Gang Yang. Cracking classifiers for evasion: a case study on the google's phishing pages filter. In *Proceedings of the 25th International Conference on World Wide Web*, pages 345–356, 2016.

[8] Rami M Mohammad, Fadi Thabtah, and Lee McCluskey. Intelligent rule-based phishing websites classification. *IET Information Security*, 8(3):153–160, 2014.

[9] Rami M Mohammad, Fadi Thabtah, and Lee McCluskey. Predicting phishing websites based on self-structuring neural network. *Neural Computing and Applications*, 25:443–458, 2014.

[10] Suhas R Sharma, Rahul Parthasarathy, and Prasad B Honnavalli. A feature selection comparative study for web phishing datasets. In *2020 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, pages 1–6. IEEE, 2020.

[11] Ke Tian, Steve TK Jan, Hang Hu, Danfeng Yao, and Gang Wang. Needle in a haystack: Tracking down elite phishing domains in the wild. In *Proceedings of the Internet Measurement Conference 2018*, pages 429–442, 2018.

[12] Bram Van Dooremaal, Pavlo Burda, Luca Allodi, and Nicola Zannone. Combining text and visual features to improve the identification of cloned webpages for early phishing detection. In *Proceedings of the 16th International Conference on Availability, Reliability and Security*, pages 1–10, 2021.

[13] Wei Wei, Qiao Ke, Jakub Nowak, Marcin Korytkowski, Rafał Scherer, and Marcin Woźniak. Accurate and fast url phishing detector: a convolutional neural network approach. *Computer Networks*, 178:107275, 2020.