Demo: PhishBench 2.0: A Versatile and Extendable Benchmarking Framework for Phishing

Victor Zeng, Xin Zhou, Shahryar Baki, and Rakesh M. Verma
University of Houston
Houston, Texas
{vzeng,xzhou21,sbaki2,rverma}@uh.edu

ABSTRACT

We describe version 2.0 of our benchmarking framework, PhishBench. With the addition of the ability to dynamically load features, metrics, and classifiers, our new and improved framework allows researchers to rapidly evaluate new features and methods for machine-learning based phishing detection. Researchers can compare under identical circumstances their contributions with numerous built-in features, ranking methods, and classifiers used in the literature with the right evaluation metrics. We will demonstrate PhishBench 2.0 and compare it against at least two other automated ML systems.

KEYWORDS

Automatic Framework, Phishing, Machine Learning, Benchmarking

ACM Reference Format:
Victor Zeng, Xin Zhou, Shahryar Baki, and Rakesh M. Verma. 2020. Demo: PhishBench 2.0: A Versatile and Extendable Benchmarking Framework for Phishing. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS '20), November 9–13, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3372297.3420017

1 INTRODUCTION

Phishing is a serious and challenging problem with a large amount of research [11]. While previous literature has used a variety of machine learning algorithms and evaluation metrics, few studies compare the performance of these algorithms using common metrics and datasets, or evaluate the metrics' suitability. Moreover, phishing researchers usually test their proposed methods against limited baselines and with few metrics when presenting new features or approach(es) [4]. Hence, there is a need for a benchmarking framework to evaluate such systems as comprehensively as possible.

PhishBench 1.0 [5, 14] is a framework for testing features and classifiers on identical pipelines. It supports both email and URL datasets and contains 12 toggleable evaluation metrics, including metrics suitable for imbalanced datasets. With configuration files, users can specify which of its numerous features and methods to use. However, its extendability is limited; researchers needed to modify the framework to add new features, classifiers, or metrics.

Besides PhishBench, there is little work on benchmarking systems for phishing. Chiew et. al. [3] prescribe a set of baseline features and a classifier (Random Forest) for benchmarking URL/website detection but do not develop a system for comparing methods against their baseline.

In this paper, we discuss PhishBench 2.0, a new and improved framework for benchmarking phishing detection systems. This framework allows researchers to easily evaluate new features and classification approaches and compare their effectiveness against existing methods from the literature. It offers evaluation metrics and methods suitable for imbalanced datasets, so researchers can test and compare their works in realistic scenarios. For baselines, it includes a rich variety of algorithms for detection or classification problems including deep learning algorithms. It has feature extraction code for over 160 features gleaned from the phishing detection literature published between 2010 and 2018 and over 90 new features from more recent work [15].

PhishBench 2.0 will be released to the public on GitHub later this year so that other researchers can also use and contribute to it. Moreover, we will continue to add new features and methods to it as our research on phishing evolves.

To summarize, our contributions are:

- We present an improved benchmarking framework for machine learning tasks, specifically classification and detection, which can dynamically load features, metrics and classifiers, and provides over 250 features, 12 classifiers, and 17 evaluation metrics.
- We implement and demonstrate the complete pipeline on datasets of phishing websites and emails from public sources.

1.1 Outline of Poster and Demonstration

Our poster will describe PhishBench 2.0 and compare it with at least two other automatic ML frameworks, selected from TPOT, H2O, Auto-WEKA and auto-sklearn. We will select these frameworks based on objective criteria such as [1, 7]. In addition, we will also demonstrate how PhishBench 2.0 can accelerate research in the phishing field by showcasing two example experiments using the framework. Besides showcasing PhishBench 2.0, our demonstration will also help the audience compare the framework with existing automated machine learning frameworks.

2 WHAT'S NEW IN PHISHBENCH 2.0

Since its inception, we have completely revamped PhishBench to significantly enhance its utility. In particular, we:

CCS '20, November 9–13, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-7089-9/20/11.
https://doi.org/10.1145/3372297.3420017

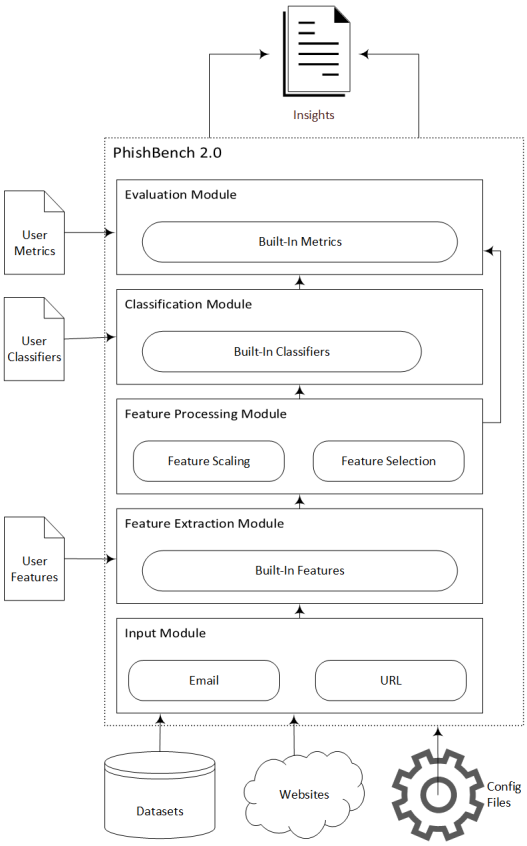

Figure 1: A high-level overview of the PhishBench 2.0 architecture

- restructured the framework as a python package, so researchers can use it as a library for custom experiments.
- rewrote the feature extraction, classification, and metrics modules using python's reflection capabilities for rapid prototyping.
- implemented an object-based data model for inputted data
- added the XGBoost [2] classifier.
- added L-tree, ranked matrix, and hidden input website features [15] (L-tree is a group of 90 fine-grained features).
- added 3 new URL features.
- implemented exporting of features and performance data for further analysis.

3 PHISHBENCH 2.0 ARCHITECTURE

PhishBench 2.0 has five modules (shown in Figure 1) that represent the different stages of a general machine learning pipeline. These modules function as independent units based on the user's needs and the input given. The framework also tracks meta-information about its modules including feature extraction time, classification time, etc.

The **Input module** loads raw email and URL datasets and parses them into a standard data model. For URL datasets, it can also download and parse the associated website and network meta-data such as DNS and WHOIS records. For email datasets, the module extracts both header information and body contents. It can parse both raw binary emails and pre-decoded text emails. If the email only contains a HTML part, it will automatically extract the text from the HTML.

The **Feature Extraction** module takes datasets from the *Input* module and extracts features. Through the configuration file, users can toggle features they wish to use. It contains over 250 built-in features. Eighty-six email features, 31 URL features and 47 network/website features come from an extensive study of phishing literature from 2010 to early 2018 [4, 5], and over 90 additional features come from more recent work [15]. In addition, users can easily implement custom features; using reflection, the module will automatically recognize and load them.

The **Feature Processing** module performs pre-processing tasks such as vectorization, feature ranking/selection, and min-max scaling on extracted features. It supports multiple popular feature selection methods, including Information Gain (IG), Gini Index (Gini), Chi-Square Metric (Chi-2), and Recursive Feature Elimination (RFE).

The **Classification** module trains classifiers on feature vectors and labels. The user can choose which classifiers to run, whether to weigh samples (if the individual classifier supports it), and whether to use default hyperparameters or perform hyperparameter tuning. It comes with 12 popular machine learning-based classifiers built-in, including classical algorithms such as Support Vector Machines and Decision Trees, and more modern algorithms such as XGBoost [2] and deep neural networks. In addition, users can define a custom classifier by implementing a scikit-learn-like [10] API.

The **Evaluation** module evaluates the performance of the classifiers using both prediction-based metrics such as accuracy and f1-score and probability-based metrics such as the ROC-AUC. It includes 17 built-in metrics, including metrics suitable for imbalanced datasets [8], and users can easily define custom metrics. For ease of processing, it reports results as a pandas [9, 13] DataFrame.

4 USING PHISHBENCH 2.0

We give a brief overview of how to use PhishBench 2.0. Upon its upcoming release, PhishBench 2.0 will come with a detailed user manual.

On a high level, the PhishBench 2.0 workflow consists of three steps: (1) implementing custom features, classifiers, and metrics, (2) creating a configuration file, and (3) performing the experiment.

To implement custom features, users simply need to write a function and decorate it with the `register_feature` decorator as in Figure 2. Depending on the feature type, PhishBench 2.0 will pass the function the appropriate data model. Metrics are implemented in a similar fashion with the `register_metric` decorator, and classifiers are implemented by extending the `BaseClassifier` abstract class.

After creating their custom components, users should generate a starter configuration file by running the `make-phishbench-config` command. Using reflection, PhishBench 2.0 will automatically recognize all classifiers, features, and metrics implemented in the current working directory and include toggles for them in the configuration file. Users then edit the configuration file to specify their datasets and the parts of the framework they wish to execute.

```python
from phishbench.feature_extraction.reflection import *
from phishbench.input import EmailHeader


@register_feature(FeatureType.EMAIL_HEADER,
'received_count')
def received_count(header: EmailHeader):
    if header.received is None:
        return 0
    return len(header.received)
```

Figure 2: A sample feature implementation for PhishBench 2.0.

In our experience, many machine-learning experiments follow a basic workflow of (1) loading a dataset from the disk, (2) extracting features, (3) pre-processing features, (4) training classifiers on a training set, and (5) evaluating the trained classifiers on a held-out test set. For performing such experiments, PhishBench 2.0 comes with a script that users can control via the configuration file. In addition, researchers who wish to perform experiments that deviate from the basic workflow can use PhishBench 2.0 as a library, wiring up the individual modules however they wish. During these custom experiments, users will still be able to control the behavior of the framework modules using configuration files.

5 DEMONSTRATIONS

We will demonstrate the versatility of PhishBench 2.0 and how it can accelerate research on phishing detection in two experiments.

5.1 Demonstration 1

First, we will show: (1) how to implement new features for PhishBench 2.0, (2) how to make a PhishBench 2.0 configuration file, and (3) how PhishBench 2.0 can accelerate experiments by evaluating the performance of features from Feng and Yue's visualization of RNNs [6] on the URL Benchmarking Dataset [12, 14]. We will implement their proposed RNN features and set up configuration files for experiments with the following feature sets:

- RNN features alone
- All built-in features
- All built-in features + RNN features

We will then run the PhishBench 2.0 basic expt. script with these configuration files and analyze the results in a Jupyter notebook.

5.2 Demonstration 2

Second, we will showcase how researchers can use PhishBench 2.0 in experiments that deviate from the basic workflow by measuring the performance scaling of various classifiers on the Email Benchmarking Dataset.

For this demonstration, we will construct an annotated Jupyter notebook using PhishBench 2.0 as a library. We will use the input, feature extraction and processing modules to extract feature vectors from the Email Benchmarking dataset [14]. Next, we will use utilities from scikit-learn to generate 4-fold splits of the features. Within each split we will use the classification module to train classifiers on subsets of 1,000 to 15,000 samples from the split's training set and the evaluation module to measure the performance of the classifiers on the validation set. Finally, we will use pandas to aggregate and plot the generated performance data.

6 CONCLUSION

We presented PhishBench 2.0, a versatile and extendable benchmarking framework for phishing detection. Through its use of reflection, the framework enables researchers to rapidly and thoroughly test new features and methods. While we used it for phishing, the full PhishBench 2.0 pipeline can also be used for other binary classification problems involving emails or URLs. Likewise, by replacing the input and feature extraction modules, researchers can repurpose the remainder of the pipeline for other binary classification tasks, e.g., threat analysis or malware detection.

ACKNOWLEDGEMENTS

We thank Radoslaw Konopka for his contributions to PhishBench 2.0. This research was supported in part by NSF grants DGE 1433817, CCF 1950297 and ARO Grant W911NF-20-1-0254.

REFERENCES

[1] Adithya Balaji and Alexander Allen. 2018. Benchmarking Automatic Machine Learning Frameworks. *arXiv preprint arXiv:1808.06492* (2018).
[2] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. New York, NY, USA, 785–794. https://doi.org/10.1145/2939672.2939785
[3] Kang Leng Chiew, Choon Lin Tan, KokSheik Wong, Kelvin S.C. Yong, and Wei King Tiong. 2019. A new hybrid ensemble feature selection framework for machine learning-based phishing detection system. *Information Sciences* 484 (2019), 153 – 166. https://doi.org/10.1016/j.ins.2019.01.064
[4] A. Das, S. Baki, A. El Aassal, R. Verma, and A. Dunbar. 2020. SoK: A Comprehensive Reexamination of Phishing Research From the Security Perspective. *IEEE Communications Surveys Tutorials* 22, 1 (2020), 671–708.
[5] A. El Aassal, S. Baki, A. Das, and R. M. Verma. 2020. An In-Depth Benchmarking and Evaluation of Phishing Detection Research for Security Needs. *IEEE Access* 8 (2020), 22170–22192.
[6] Tao Feng and Chuan Yue. 2020. Visualizing and Interpreting RNN Models in URL-based Phishing Detection. In *Proc. 25th ACM Symposium on Access Control Models and Technologies*. 12. https://doi.org/10.1145/3381991.3395602
[7] P. J. A. Gijsbers, Erin LeDell, Janek Thomas, Sébastien Poirier, Bernd Bischl, and Joaquin Vanschoren. 2019. An Open Source AutoML Benchmark. *ArXiv* abs/1907.00909 (2019), 8.
[8] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. 2017. Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning. *Journal of Machine Learning Research* 18, 17 (2017), 1–5. http://jmlr.org/papers/v18/16-365
[9] The pandas development team. 2020. pandas-dev/pandas: Pandas. https://doi.org/10.5281/zenodo.3509134
[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.
[11] Rakesh M. Verma and David J. Marchette. 2019. *Cybersecurity Analytics*. CRC Press, Boca Raton, FL.
[12] Rakesh M. Verma, Victor Zeng, and Houtan Faridi. 2019. Data Quality for Security Challenges: Case Studies of Phishing, Malware and Intrusion Detection Datasets. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. New York, NY, USA, 2605–2607.
[13] Wes McKinney. 2010. Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference*, Stéfan van der Walt and Jarrod Millman (Eds.). 56 – 61. https://doi.org/10.25080/Majora-92bf1922-00a
[14] Victor Zeng, Shahryar Baki, Ayman El Aassal, Rakesh Verma, Luis Felipe Teixeira De Moraes, and Avisha Das. 2020. Diverse Datasets and a Customizable Benchmarking Framework for Phishing. In *Proceedings of the Sixth International Workshop on Security and Privacy Analytics (IWSPA '20)*. Association for Computing Machinery, New York, NY, USA, 35–41. https://doi.org/10.1145/3375708.3380313
[15] Xin Zhou and Rakesh Verma. 2020. Phishing Sites Detection from a Web Developer's Perspective Using Machine Learning. In *Proc. of the 53rd Hawaii Int'l Conf. on System Sciences (HICSS)*. 10. https://doi.org/10.24251/HICSS.2020.794