Fingerprinting Website

Wenchange Liu
wchliu@ucdavis.edu
University of California, Davis
Davis, California, USA

Ziyang Chen
zynchen@ucdavis.edu
University of California, Davis
Davis, California, USA

Jinxiao Yu
jnxyu@ucdavis.edu
University of California, Davis
Davis, California, USA

Abstract—**The browsing history contains highly sensitive information about users. Several tools, such as the Tor network, have been developed to encrypt the network traffic to protect users' online privacy. However, some past research indicates that encrypting traffic is not sufficient for protecting the users' privacy. Observable patterns in the metadata of encrypted traffic may reveal the web page that the user is visiting. In this work, we analyze the browser's microarchitectural footprint by collecting information about hardware performance counters with the help of Machine Learning and Deep Neural Networks. We have collected profiling data from about 100 websites considered both the close-world and open-world scenario. And we are able to achieve the accuracy in the close-world dataset by up to 84.7% and the open-world dataset by up to 84.2%. The results show that hardware performance counters can clearly infer users' activities and reveal users' privacy.**

I. Introduction

Over the last two decades, with the development of the Internet, Web applications have become an indispensable component of our lives. The accessibility of the Internet allows attackers to obtain malicious tools from relevant forums and launch attacks with simple techniques and cheap costs. In recent years, attackers have used the corresponding web vulnerability loophole to attack based on websites fingerprints, which have caused data breaches and user privacy leaks. According to the Identity Theft Resource Center's 2021 Data Breach Report, there were 1,862 data breaches last year, surpassing 2020's total of 1,108 and the previous record of 1,506 set in 2017. [7] The personal data breach could reveal private or sensitive information about them, including sexual orientation or political beliefs and affiliations. This information allows attackers to threaten users based on their information to achieve profit or other purposes. Therefore, it is significant to accurately identify the fingerprint information of websites, discover the possible security vulnerabilities to provide a robotic website and protect users' privacy.

While users browse websites, there are special pieces of information left by web service components during development that can identify their types and version, we call these web fingerprints. Web fingerprinting refers to identifying relevant information using the existing web fingerprint library or the Machine Learning algorithm. Web fingerprinting technology can be used in the data collection process of web penetration. Accurately identifying the information of web service components can effectively detect its existing security loophole. In reality, the majority of web attackers may focus on utilizing the existing security weaknesses of websites and use corresponding attack methods to obtain advanced permission and sensitive data. In this work, we mainly focus on fingerprinting the websites the users are opening based on hardware events, thus simulating the fingerprinting techniques to study the privacy attack method.

In the web attack method, we summarize the following points from three groups of people:

- **From the attacker's perspective**: their top priority is to obtain valid information for implementing an attack. So the web vulnerability detection needs first to identify the type and version of the webserver and implement proper exploitation methods based on different machines.
- **From the victim's perspective**: raise awareness of cybersecurity, use secure browsers to access websites, and install relevant security software on computers and smartphones. When registering accounts on different websites, do not use the same password, especially a weak password. Always be cautious of unsafe websites, and monitor whether you have entered the correct website by carefully observing the URL address.
- **From the defender's perspective**: complete web application penetration testing. Web application penetration testing is done by simulating unauthorized attacks internally or externally to access sensitive data. Web penetration helps end-users find out the possibility for a hacker to access data from the Internet, find out the security of their email servers, and get to know how secure the web hosting site and server are. [8] One of the first tasks when conducting a web application penetration test is to identify the version of the webserver and the web application. That is because it allows developers to discover all the well-known vulnerabilities that are affecting the web server and the application. This process is called web fingerprints, as we mentioned above. [9] On the protection strategy based on data analysis, multiple types of data (vulnerability information, threat intelligence) are selected for correlation analysis based on the fingerprint information of target websites.

For attackers, the most straightforward method to track a device is by using HTTP cookies. However, this is just one of the web fingerprinting methods. In scientific research, web fingerprints are mainly divided into the following types: compare the MD5 of a specific file, scan the keywords web pages, keyword matching for request header information, identify keywords contained in partial URLs, detect user activity characteristics based on hardware activity, etc. Within the methods of web fingerprints, most of them require the attackers to inject JavaScript code via the website to the victim's browser. Furthermore, an overlay network (Tor) can protect users by disabling features used to track users. While browsers have well-defined privacy assurances, they are still far from perfect. Nowadays, an attacker can infer web browsing activity by exploiting microarchitectural leakages at the hardware level. [10] In 2012, Jana and Shmatikov [11] found they can detect opened websites using memory footprints of processes. In 2015, Liu et al. [12] addressed the entire last-level cache of a processor that can be profiled and infer a small set of opened websites.

This work shows how to infer private user activity by analyzing Hardware Performance Counters (HPC). We follow the data collection process from Leaked-web [1] and implement our own Deep Neural Network (DNN) model, we demonstrate that it is possible to infer user's activity by either using a close-world dataset or an open-world dataset. For the experiments in this work, we use perf functionality in the Linux kernel to profile the top websites from all over the world to properly construct the database, which is using features as labels. The recent Machine Learning algorithms provide us the ability to classify complex easily, noisy data in an effective manner. We utilize wide-known open-source software Weka [18] to perform traditional Machine Learning (ML) methods, such as Random Forest, Multiclass Classifier, and Filtered Classifier. Furthermore, we choose Multi-Layer Perceptron and LSTM as our DNN models. After we perform all the experiments, we are able to achieve better results than our predecessor Leaked-Web.

Our contribution. In summary, we

- use Perf to profile top websites from all over the world and collect hardware events,
- build a database of hardware events as features and websites as labels,
- cover top 30 websites from top Alexa sites and 70 more top websites ranked by different countries,
- utilize Weka to perform machine learning-based website fingerprinting models for identifying the target website,
- implement DNN models for target websites,
- analyze data consistency and models accuracy,
- summarize critical characteristics of fingerprinting websites using hardware events to prevent privacy leakage,
- we open-sourced our source code at https://github.com/williamlwclwc/EEC270-Fingerprinting-Websites

The rest of the paper is organized as follows. Section II provides background information and related work for HPC, ML algorithms, and DNN models. Section III shows our implementation and section IV evaluates our results. Finally, section V concludes this work and proposes some future works.

II. Background and Related Work

This section provides background information and related work regarding HPC, ML algorithms, and DNN models.

A. Hardware Performance Counters (HPC)

Performance counters are CPU hardware registers that count hardware counters such as instructions executed, cache-misses suffered, or branches mispredicted. They form a basis for profiling applications to trace dynamic control flow and identify hotspots. Perf provides rich generalized abstractions over hardware-specific capabilities. Among others, it provides per task, per CPU, and per-workload counters, sampling on top of these, and source code event annotation.

Tracepoints are instrumentation points placed at logical locations in code, such as for system calls, TCP/IP events, file system operations, etc. When not in use, these have negligible overhead and can be enabled by the perf command to collect information, including timestamps and stack traces. Perf can also dynamically create tracepoints using the kprobes and uprobes frameworks for kernel and userspace dynamic tracing. perf began as a tool for using the performance counters subsystem in Linux and has had various enhancements to add tracing capabilities. [2]

B. Machine Learning (ML) Algorithm

We carefully select some of the most feasible options from Weka's functionality in this work. Then, based on Leaked-web and other scientific material, we choose the following three Machine Learning algorithms: Random Forest, Multiclass Classifier, and Filtered Classifier. Here are the definitions of them.

1) Random Forest: The random forest is a classification algorithm consisting of many decisions trees. Decision Trees are used to classify samples by creating branches for the given data features. Random forest uses bagging and features randomness when building each individual tree to try to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree.

2) Multiclass Classifier: It is also known as Multiclass Classification. Multiclass classification is a classification task with more than two classes. Each sample can only be labeled as one class. For example, the classification uses features extracted from a set of fruit images, where each image may either be of an orange, an apple, or a pear. Each image is one sample and is labeled as one of the three possible classes. Multiclass classification assumes that each sample is assigned to one and only one label - one sample cannot, for example, be both pear and an apple.

3) Filtered Classifier: This is a class in Weka for running an arbitrary classifier on data that has been passed through an arbitrary filter. Like the classifier, the structure of the filter is based exclusively on the training data and test instances will be processed by the filter without changing their structure. If unequal instance weights or attribute weights are present, and the filter or the classifier cannot deal with them, the instances or attributes are resampled with replacement based on the weights before they are passed to the filter or the classifier (as appropriate).

C. Deep neural network (DNN)

We choose Multi-layer perceptron and Long Short-Term Memory for the DNN model to classify the selected websites.

1) Multi-layer perceptron: Multi-layer perceptron (MLP) is a supplement of a feed-forward neural network. It consists of three types of layers—the input layer, output layer, and hidden layer. The input layer receives the input signal to be processed. The output layer performs the required task such as prediction and classification. An arbitrary number of hidden layers placed between the input and output layer are the actual computational engine of the MLP. [13] MLPs are designed to approximate any continuous function and can solve problems that are not linearly separable. The major use cases of MLP are pattern classification, recognition, prediction, and approximation.

2) Long Short-Term Memory (LSTM): Long Short-Term Memory (LSTM) networks are a type of recurrent neural network capable of learning order dependence in sequence prediction problems. [14] This behavior is required in complex problem domains like machine translation, speech recognition, and more. LSTMs are a complex area of deep learning. It can be hard to get your hands around what LSTMs are, and how terms like bidirectional and sequence-to-sequence relate to the field.

III. Implementation

A. Experiment Environment

Our experiment is conducted using an x86 Arch Linux laptop machine with an Intel i7 CPU, Linux kernel version 5.16.11, 8G DRAM, and with Google Chrome stable version 98.0.4758.102 installed. We used Weka to train the traditional machine learning models locally and PyTorch to train deep neural networks on Google Colab. The following table includes the configuration of our machine:

TABLE I
Experiment Environment.

Linux Kernel	5.16.11
OS	Arch Linux x86_64
CPU	Intel i7-7500U
DRAM	8G
Browser	Google Chrome 98.0.4758.102

B. Overview of the Fingerprinting Model

Our task is to fingerprint target websites via hardware events. Regarding this task as a machine learning problem, we are classifying websites given hardware events as features, i.e. the machine learning model takes in a set of features of a website, and then outputs which website it is.

To do this using a supervised machine learning approach, we will need to profile websites data using Perf [2] first, each data entry consists of a set of hardware events features, and a ground truth label indicating which website it is. Then we can split the database into a training set and a testing set where we can train the model on the training set and then test it on the testing set. We can calculate the accuracy of a model by comparing the output generated label of our model with the ground truth label of our test data: $Accuracy = matchedSamples/allTestingSamples$

C. Generating Dataset

1) Dataset Overview: In this work, we use Perf [2] to collect the hardware events features when we open the website with a browser. Perf is a profiling and performance analysis tool that can help to track the hardware performance counters. As introduced in Section I, we use this tool to monitor the HPC, i.e. hardware events counts each second.

The more events we monitor, the more data would be available for our models to learn, however, also more workload for data collection and training. Thus, we want to select not all, but a set of effective hardware events as our features. Here we followed the conclusion from paper Leaked-Web [1]: they conducted an attributes ranking, and we select the top 4 events of that ranking: cache-misses, node-loads, branch-misses, branch-load-misses as the features. Similarly, faster sample rate and more profiling time generates more features, but more workload as well. Here we also followed the Leaked-Web [1] paper's conclusion and choose a sample interval of 1s and to profile 5s for each trace.

We call each data entry a trace of a website. It contains the 4 events counts in the first 5s when we opened a website with a browser, i.e. 20 features, and a label ranging from 1 to 30 indicating which website it is, which means, each trace has 20 features and 1 label.

In machine learning, one data entry for one category usually is not enough to train an effective classification model. So in our experiment, we collected 70 traces for each class, 50 for training, and 20 for testing. In total, we have 2100 traces of data, 1500 for training, and 600 for testing.

2) Automatic Data Collection: As introduced in the previous section, we have need 2100 traces in total, which means opening websites and executing Perf to monitor them 2100 times. It would be tiresome for individuals if we do it manually. And that is why implementing an automatic data collection pipeline using scripts is necessary.

The core command is to call Perf to execute the browser with an URL for a certain duration of time, we can use:

```
 timeout <duration(s)> perf stat -e
<events> -I <sample interval> -o <output
file name> google-chrome-stable <URL>
```

to generate one trace of data in a txt file, named after its category label and its number(range from 1 to 70) of the same class of data. To automatically generate 2100 traces, we wrote the Python script with 2 loops, in the outter loop, we went through all 30 target websites, and in the inner loop, we repeatedly call the Perf command 70 times, in this way, we get the data in 2100 files.

Next, we wrote another script to iterate each txt file, parse the data and make it one line in the CSV file. Then we append the label and insert it into training CSV or testing CSV according to the txt file filename. In this way, we got a training CSV file with $30*50 = 1500$ traces, a testing CSV file with $30*20 = 600$ traces, and each trace has 20 columns of features and 1 label.

3) Open World vs Closed World: The previous discussions are based on the closed world assumption, where we only consider the 30 target websites. However, in reality, the users may open websites other than our 30 targets, and we have to make sure that our proposed model does not classify those websites as our targets. To modify our previous closed world dataset into an open world dataset, we will need to:

1) Introduce a new category as ”non-sensitive” websites. We only need to know they are not one of our 30 targets, we do not care which website they exactly are.
2) Include another 70 websites traces into our dataset, we still use 50 traces for training, and 20 traces for testing. However, this time, we have 70 traces from 70 different websites(we select top 30-100 websites from Alexa Top Websites).
3) For any classifier models, we need to change the output classes from 30 to 31.

D. Traditional Machine Learning Models

We use 3 traditional machine learning methods to model the website fingerprinting attack. The implementation stage consists of a building step and attack step [1].

In the building step, multiple traces from each website are collected and labeled. The labeled dataset is used to train traditional machine learning models. In the attacking step, the proposed attack model receives unlabeled traces in which each of the traces is corresponding to a user's website visit and the trained classifier outputs the prediction results. The rationale for choosing these models is that they are from different branches of machine learning tasks compared to deep learning models and can provide baselines for our later deep learning results.

1) Experimental set up in Weka: We implement these 3 models using Weka, a machine learning tool that contains a collection of visualization tools and algorithms for data analysis and predictive modeling. For data pre-processing, we first standardize the data to avoid large range weight problems, and then we normalize the value of the numeric column of “label” without distorting differences in the ranges of values to perform the classification tasks. As mentioned in Section III-C, our data considers 20 features, have 50 traces per website for training set (1500 samples in total) and 20 traces per websites for testing set (600 in total). Additionally, we consider the open world dataset (a set of non-sensitive websites), which contains additional 70 traces in which each of them represents a single website's behavior.

2) Implemented Models: (a) Multiclass Classifier: We first use a common machine learning task: Multiclass Classification. Multiclass classification is the problem of classifying instances into one of three or more classes. It aims at producing a learning model from a labeled training set and has many approaches like neural networks, decision trees, one-versus-all decomposition, etc. [4] The Multiclass classifier in Weka is a 2-class classifier. If the base classifier cannot handle instance weights, and the instance weights are not uniform, the data will be resampled with replacement based on the weights before being passed to the base classifier. And we use one-against-all method.

(b) Filtered Classifier: We then implement the FilteredClassifier in Weka, which is running an arbitrary classifier on data that has been passed through an arbitrary filter. The structure of the filter in this model is based exclusively on the training dataset, while the testing data will be processed by the filter without making any changes to the structure. For the FilteredClassifier in Weka, if the filter or the classifier are not able to handle the unequal instance weights or attribute weights presented, the instances or attributes will resampled with replacement based on the weights prior to their pass to the filter.

(c) Random Forest: For the last traditional model, we implement the Random Forest model. Random forest is a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees [6]. It is a Supervised Classification method that is applied to test data set where the trees are constructed while the resultant individuals are combined to predict the class label [5].

E. Deep Learning Models

In this work, all the deep neural network models are built using PyTorch [15]. PyTorch not only makes it easy for users to build and train deep learning models by providing various built-in layers and computation methods but also allows users to customize the network architecture and define how the data goes forward and backward. We think it can balance simplicity and flexibility.

For deep neural network models, there are a lot of factors that can affect the final results, such as hyperparameters, initialization, architectures. Due to time limitations, we haven't conducted a comprehensive hyperparameters search to find an optimal parameters configuration or testing with all kinds of initialization and architecture designs, but we still tried several combinations of them to achieve decent accuracies and we found tuning the parameters will have limited impact on a already high-enough accuracy.

We also applied standardization during preprocessing to achieve better performance. We use the ”standardscaler” method in the scikit-learn [17] package to calculate the standardization metric using the training set and then apply it to the testing set so that we make sure we are not ”cheating” by manipulating the testing set.

1) Multi-layer Perceptron(MLP) Model: First, we tried the simplest deep neural network: Multi-layer Perception(MLP). Our implementation is a little bit different than the traditional MLP network, we adapt it to a more modern setting. It is also a sequential model whose main components are linear layer, activation functions, and softmax, however, we used more hidden layers to enhance the representational capability of the deep neural network. And since our dataset is comparatively small for deep learning, and we found the training accuracy can be much higher than testing accuracy, therefore we also used a dropout layer to generalize it and avoid the overfitting issue. We used relu for activation functions, the Adam optimizer, and cross entropy loss function for our multi-class classification task.

The architecture of our MLP for closed world setting is:

```python
class MLP(nn.Module):

    def __init__(self):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(20, 48)
        self.fc2 = nn.Linear(48, 96)
        self.fc3 = nn.Linear(96, 128)
        self.fc4 = nn.Linear(128, 30)
        self.dropout = nn.Dropout(0.1)

    def forward(self, x):
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
        x = F.relu(x)
        x = self.fc3(x)
        x = F.relu(x)
        x = self.dropout(self.fc4(x))

        return x
```

And for open world setting, we change the output classes number from 30 to 31:

```python
class MLP(nn.Module):

    def __init__(self):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(20, 48)
        self.fc2 = nn.Linear(48, 96)
        self.fc3 = nn.Linear(96, 128)
        self.fc4 = nn.Linear(128, 31)
        self.dropout = nn.Dropout(0.1)

    def forward(self, x):
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
        x = F.relu(x)
        x = self.fc3(x)
        x = F.relu(x)
        x = self.dropout(self.fc4(x))

        return x
```

Note that we haven't implemented a softmax function at the output layer, because the softmax has been included in the cross entropy loss function. And the softmax function will not affect the final prediction result during evaluation on the testing set.

2) Long Short-Term Memory(LSTM) Model: As introduced in section II, the long short-term memory(LSTM) module is famous for solving sequence-related problems. In our case, the input features have a time order, which is kind of a sequence data, so we suppose LSTM can help with it. To implement a LSTM model, we simply replaced the first Linear layer of our MLP model with the LSTM module, removed a hidden layer for simplicity, and the architecture of our dnn for closed world setting is like this:

```python
class LSTM(nn.Module):

    def __init__(self):
        super(LSTM, self).__init__()
        self.fc1 = nn.LSTM(20, 48)
        self.fc2 = nn.Linear(48, 96)
        self.fc3 = nn.Linear(96, 30)
        self.dropout = nn.Dropout(0.1)

    def forward(self, x):
        x, _ = self.fc1(x.view(len(x), 1, -1))
        x = self.fc2(x.view(len(x), 48))
        x = F.relu(x)
        x = self.dropout(self.fc3(x))

        return x
```

Similarly, we change the number of output classes from 30 to 31 for open world setting:

```python
class LSTM(nn.Module):

    def __init__(self):
        super(LSTM, self).__init__()
        self.fc1 = nn.LSTM(20, 48)
        self.fc2 = nn.Linear(48, 96)
        self.fc3 = nn.Linear(96, 31)
        self.dropout = nn.Dropout(0.1)

    def forward(self, x):
        x, _ = self.fc1(x.view(len(x), 1, -1))
        x = self.fc2(x.view(len(x), 48))
        x = F.relu(x)
        x = self.dropout(self.fc3(x))

        return x
```

IV. Evaluation

A. Experiment Overview

We investigate the inference of opened websites in a distinct scenario, Google Chrome on Intel, hosted on Arch Linux system. From [1], we know that it is necessary to identify the most important HPCs for classifying the websites. As a result, four kinds of HPC features are measured using Perf to simulate the malicious attack, namely Cache-misses, Node-loads, Branch-misses, Branch-load-misses. As mentioned in Section III-C, each trace contains the 4 events counts in the first 5s when we opened a website with a browser, i.e. 20 features. And we monitor HPEs for 30 of the most visited websites globally [3].

During the online malicious attack implementation phase, close world and open world data are both taken into consideration. We implement 5 machine learning methods to train models and find the most effective one, which contains 3 traditional machine learning models and 2 deep learning models. Additionally, the effectiveness of "standardscaler" is also evaluated. About 70% of the data is assigned to training set (50 traces per websites) and 30% of data is assigned to testing set (20 traces per websites). In this section, the effectiveness of different machine learning models is evaluated comprehensively in terms of classification accuracy. Such evaluation gives insight into the cost of HPCs-based websites fingerprinting attacks and further indicate the necessity of protection approaches.The overall comparison of the model results are shown on the table below.

B. ML Classification Models Comparsion

We use 3 traditional machine learning methods to model the website fingerprinting attack. The rationale for choosing these machine learning models is that they are commonly used in real world application, and they are from different branches of machine learning compared to deep learning models so that they can provide baselines for our later deep learning model results. Moreover, 2 deep learning machine learning methods are used to model the website fingerprinting attack, namely

TABLE II
EVALUATION RESULTS.

Model	RF	MultiClass	Filtered	MLP	MLP (standardized)	LSTM (standardized)
Accuracy (close world)	81.7%	78.4%	65.0%	77.7%	84.7%	84.5%
Accuracy (open world)	80.5%	77.5%	62.4%	75.8%	82.1%	84.2%

Multi-layer perceptron (MLP) and Long Short-Term Memory (LSTM). We implement a simplest deep neural network MLP and a time series based networks LSTM to compare the effectiveness of them.

TABLE III
MULTICLASS CLASSIFIER PARAMETERS.

	batchSize	method	numDecimalPlaces	randomWidthFactor	seed
MultiClass Classifier	100	1-against-all	2	2.0	1

TABLE IV
RANDOM FOREST PARAMETERS.

	batchSize	maxDepth	numDecimalPlaces	numExecutionSlots	numIterations	seed
Random Forest	100	0	2	1	100	1

TABLE V
FILTERED CLASSIFIER PARAMETERS.

	batchSize	filter	numDecimalPlaces	resume	seed
Filtered Classifier	100	Discretize	2	false	1

1) Traditional ML models evaluation: The traditional model parameters set up in Weka are shown on the above table III, IV, V.

The dataset is both standardized and partially normalized. As observed from the results, among the traditional models we considered, the Random Forest model achieve the greatest classification accuracy, while the Filtered Classifier model gain the lowest success rate.

In particular, the classification accuracy for 50 training observations per website of Random Forest is 81.7%, for it includes multiple classifiers to predict class label values with past data set and is efficient in handling thousands of data inputs. This indicates that such Random Forest based fingerprinting attack can be relatively effective in inferring users' browsing history.

For Filtered Classifier, the classification accuracy for 50 training observations per website is 65%, which can be considered that this model can not efficiently perform the classification and its effectiveness for website fingerprinting attack is little under this amount of input sample.

Moreover, for Multiclass classifier, the classification accuracy for 50 training observations per website is 78.4%, which is relatively well, for it is capable of applying error-correcting output codes for increased accuracy.

Upon applying the additional open world dataset (70 traces from 70 unique websites), a slight drop of classification accuracy can be observed from the result for all traditional machine learning models. The classification accuracy of the Random Forest model dropped to 80.5%, the accuracy of the Multiclass Classifier model dropped to 77.5% and the accuracy of the Filtered Classifier model dropped to 62.4%. This indicates that the open world non-sensitive web pages data affect the effectiveness of website fingerprinting attack for decreasing the prediction accuracy.

2) Deep Learning models evaluation: The deep learning model results are also shown in the table II.

We investigate results with both a standardized and a none standardized dataset of the MLP model, and the result with a standardized dataset of the LSTM model. The results are obtained with 5000 epochs of training with a learning rate ranging from $1e-3$ to $1e-5$.

As observed from the results, the MLP model with standardized inputs achieved the highest classification accuracy among the models using closed world dataset, and the LSTM model with standardized inputs reach the highest classification accuracy among the models using open world dataset.

To be specific, the classification accuracy for 50 training observations per website of MLP with none standardized inputs is 77.7%, and the classification accuracy for 50 training observations per website of MLP with standardized inputs reaches 84.7%.

The reason for standardization is that if one feature has very large values, it will dominate over other features when calculating the distance, and standardization gives all features the same influence on the distance metric. And as we can conclude from the results that after standardizing features by removing the mean and scaling to unit variance, the classification accuracy of the MLP model increased significantly and reached the highest accuracy among the models using the closed world dataset in our experiment. That also indicates that MLP based fingerprinting attacks can be effective in accurately predicting users' browsing history at run-time with several hardware features of Intel processor, for MLP models works well with large input data, can be used to approximate any continuous function and be applied to complex non-linear problems.

For the LSTM model, the classification accuracy for 50 training observations per website is 84.5%, which is very close to our greatest value of the MLP model. This indicates that LSTM based fingerprinting attacks can also have great performance in inferring users' browsing history with hardware events.

Moreover, upon applying the additional open world dataset (70 traces for single unique website), the classification accuracy of MLP model with none standardized dataset dropped to

75.8%, the accuracy of MLP model with standardized dataset dropped to 82.1%, and the accuracy of LSTM model with standardized dataset dropped to 84.2%.

V. Conclusion and Future Works

In conclusion, we have introduced the background, the related works, and the significance of the fingerprinting website problem. We explained how we collect data and build a database in both the closed world and open world scenarios. Then we introduced typical machine learning and deep learning approaches for classification problems, how we implemented the models, and finally evaluated their classification accuracies on the testing dataset to prove that machine learning based approaches is effective for the fingerprinting websites.

Though we achieved decent accuracy using random forest and DNN models on both closed world and open world dataset, we think the following future works may further improve the results:

1) More data and more features: Due to time limitations, we only include 70 traces for each target website and only 20 features for each trace. However, usually for machine learning, especially deep learning, the models prefer more features and more data so that those models can reach their full potential.
2) Hyperparameters search: There are a lot of hyperparameters for deep neural networks including how many hidden layers, learning rate, initialization, dropout ratio, etc. In this work, we kind of blindly tested a few combinations, however, there are more systematic ways to explore the hyperparameters such as using AutoML approaches or simply doing a grid search.
3) Exploring more advanced DNN architectures: In this work, we tested MLP and LSTM deep neural networks. Nevertheless, the more advanced architectures like the Transformers [16] have been dominating the deep learning domain, maybe a more complex and advanced architecture can let our model reach higher accuracies.

References

[1] Wang, H. (2021). Leaked-Web: Accurate and Efficient Machine Learning-Based Website Fingerprinting Attack through Hardware Performance Counters. ArXiv:2110.01202 [Cs]. http://arxiv.org/abs/2110.01202

[2] Perf. In https://perf.wiki.kernel.org/index.php/Main Page.

[3] ALEXA INTERNET INC. The top 500 sites on the web, 2022. https://www.alexa.com/topsites

[4] Neha Mehra and Surendra Gupta. (2005). Survey on Multiclass Classification Methods

[5] Gashler M, Carrier CG, Martinez T (2008) Decision tree ensemble: small heterogeneous is better than large homogeneous. In: Proceedings of 7th international conference on machine learning and applications, pp 900–905

[6] Breiman, L. Random Forests. Machine Learning 45, 5–32 (2001). https://doi.org/10.1023/A:1010933404324

[7] Fowler, B. (2022, January 24). Data breaches break record in 2021. CNET. Retrieved March 15, 2022, from https://www.cnet.com/news/privacy/record-number-of-data-breaches-reported-in-2021-new-report-says/

[8] Karande, Manohar. “Beginners Guide to Web Application Penetration Testing.” Software Testing Help, 3 Mar. 2022, https://www.softwaretestinghelp.com/getting-started-with-web-application-penetration-testing/.

[9] Administrator., PENETRATION TESTING LAB. (2012, August 1). Web application fingerprinting. Penetration Testing Lab. Retrieved March 15, 2022, from https://pentestlab.blog/2012/08/01/web-application-fingerprinting/

[10] Berk Gulmezoglu, Andreas Zankl, Thomas Eisenbarth, Berk Sunar: “PerfWeb: How to Violate Web Privacy with Hardware Performance Events”, 2017; [http://arxiv.org/abs/1705.04437 arXiv:1705.04437]

[11] JANA, S., AND SHMATIKOV, V. Memento: Learning secrets from process footprints. In Security and Privacy (SP), 2012 IEEE Symposium on (2012), IEEE, pp. 143–157.

[12] LIU, F., YAROM, Y., GE, Q., HEISER, G., AND LEE, R. B. Last-level cache side-channel attacks are practical. In Security and Privacy (SP), 2015 IEEE Symposium on (2015), IEEE, pp. 605–622.

[13] S. Abirami, P. Chitra, Chapter Fourteen - Energy-efficient edge based real-time healthcare support system, Editor(s): Pethuru Raj, Preetha Evangeline, Advances in Computers, Elsevier, Volume 117, Issue 1, 2020, Pages 339-368, ISSN 0065-2458, ISBN 9780128187562, https://doi.org/10.1016/bs.adcom.2019.09.007.

[14] Brownlee, J. (2021, July 6). A Gentle Introduction to Long Short-Term Memory Networks by the Experts. Machine Learning Mastery. https://machinelearningmastery.com/gentle-introduction-long-short-term-memory-networks-experts/

[15] Facebook Inc. PyTorch. https://pytorch.org/

[16] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention Is All You Need. ArXiv:1706.03762 [Cs]. http://arxiv.org/abs/1706.03762

[17] Pedregosa et al. Scikit-learn: Machine Learning in Python, JMLR 12, pp. 2825-2830, 2011.

[18] Eibe Frank, Mark A. Hall, and Ian H. Witten (2016). The WEKA Workbench. Online Appendix for ”Data Mining: Practical Machine Learning Tools and Techniques”, Morgan Kaufmann, Fourth Edition, 2016.