URL-based Web Tracking Detection Using Deep Learning

Ismael Castell-Uroz*, Théo Poissonnier†, Pierre Manneback† and Pere Barlet-Ros*
*Department of Computer Architecture, Universitat Politècnica de Catalunya
Barcelona, Spain
Email: {icastell, pbarlet}@ac.upc.edu
†Computer Science Unit, Faculty of Engineering, University of Mons
Mons, Belgium
Email: theo.poissonnier@student.umons.ac.be / pierre.manneback@umons.ac.be

***Abstract*—The pervasiveness of online web tracking poses a constant threat to the privacy of Internet users. Millions of users currently employ content-blockers in their web browsers to block tracking resources in real time. Although content-blockers are based on blacklists, which are known to be difficult to maintain and easy to evade, the research community has not succeeded in replacing them with better alternatives yet. Most of the methods recently proposed in the literature obtain good detection accuracy, but at the expense of increasing their complexity and making them more difficult to maintain and configure by the end user. In this paper, we present a new web tracking detection method, called Deep Tracking Detector (DTD), that analyzes the properties of URL strings to detect tracking resources, without using any other external features. Consequently, DTD can easily be implemented in a browser plugin and operate in real time. Our experimental results, with more than 5M HTTP requests from 100K websites, show that DTD achieves a detection accuracy higher than 97% by looking only at the URL of the resources.**

***Index Terms*—web tracking, deep learning, URL classification**

I. Introduction

Several studies (e.g. [1], [2]) have shown that, nowadays, more than 70% of the most popular web services have tracking systems running in the background. Web tracking allow online companies to collect information about their users to improve their services. For instance, web tracking is the basis of many search customization algorithms and targeted advertisement campaigns. However, from the user perspective, this is achieved at the expense of their own privacy. Some studies found the collected data to be used for many other purposes, such as price discrimination [3], assessment of financial credibility [4] or determination of insurance coverage [5]. Moreover, the appearance of the so-called *third-party trackers* and *data-brokers* aggravated the situation. The former are services with the ability to centralise personal information from multiple different domains, while the latter are companies willing to collect personal information to sell it to other companies. According to [6], the web analytics market size was $2.63B in 2018 and it is expected to reach $10.73B by 2026.

Although the research community has dedicated great efforts to improve the privacy protection measures by means of new complex techniques (e.g. [7]–[9]), the most popular approach to block web tracking systems still relies on traditional content-blockers [10]; browser plugins that block URLs found on manually-curated pattern lists. The reason behind this is that latest methods proposed in the literature are difficult to adopt by common users. For example, recent proposals require users to install instrumented versions of web browsers [7], get measurements from multiple vantage points [11] or share privacy-sensitive information, like cookies [12].

In this work, we present Deep Tracking Detector (DTD), a novel approach that uses deep learning to automatically detect tracking URLs. The classification is done locally and exclusively based on the properties of the URL string, without the necessity of using any external features or communications. This approach has several advantages: (i) it does not require to download the source files, so malicious resources can be blocked in advance, (ii) it is more robust against the use of minification and code obfuscation techniques [13], (iii) it is more difficult to evade than traditional content-blockers, (iv) it can potentially generalize to new tracking domains, which would be missed by traditional content-blockers based on blacklists, and (v) it is lightweight enough to be included in a browser plugin.

In order to evaluate the performance of DTD, we collected over 5M HTTP requests from the top 100K most popular websites as per the Alexa's list [14]. The corresponding URLs were labeled using the most popular pattern lists currently available. Our experimental results show that DTD obtains a detection accuracy over 97% using only the characters of the URL as input. This result indicates that tracking URLs actually have distinguishable properties that can be effectively exploited for web tracking detection.

The rest of the paper is organized as follows. Section II revises the most relevant related work in this area. In Section III we present the architecture of our deep learning detector, while the evaluation is presented in Section IV. Lastly, Section V concludes the paper and presents the future work.

II. Related work

In recent years, there have been many research studies that focused on improving online privacy. We refer the interested

reader to [15] for a comprehensive survey on the existing web tracking mechanisms and the available protection measures. Regarding tracking detection methods, Ikram et al. in [9] and Wu et al. in [8] proposed machine learning algorithms based on the analysis of JavaScript code syntax to detect tracking patterns. Other works, such as Iqbal et al. in [7], proposed complex combinations of JavaScript code attribution, DOM code inspection and network requests annotation to detect tracking pattern signatures. However, most of these approaches are very specialized and complex, and usually require major modifications to the browsers, fact that prevent common users to adopt these solutions.

In [12] Metwalley et al. present a system to find unique identifiers embedded in the HTTP headers or HTTP requests. To this end, they compare the requests of different users accessing the same domain to discover parameters that can be used as identifiers. Similarly, Wu et al. present in [16] a system to detect the same identifiers collaboratively by means of a k-anonimity machine learning algorithm. Both approaches use only the URLs, similarly to our work. However, they need to compare requests from different users to detect identifiers. The main advantage of this approach is that it is easy to deploy using a browser plugin that communicates to a centralized system responsible for the comparison. The main drawbacks are the privacy implications involved in sending all the HTTP requests done by the user to make the comparison, and the delay introduced by this communication.

In [17] Le et al. propose a deep learning method to detect malicious and phishing websites. The system applies a combination of two Convolutional Neural Networks over the characters and words of the URL to find patterns that could identify malicious websites. In contrast, we show that a simpler and more compact architecture can work more effectively in the case of web tracking detection.

On the other hand, content-blockers such as *AdBlock Plus* [18], *Ghostery* [19] or *uBlock Origin* [20] compare the URLs to a predefined list of patterns indicating the domains to be blocked. These solutions are simple and easy to deploy, as they only require local access to the URL. EasyList [21] and EasyPrivacy [22] are the most popular and precise lists currently available. Both of them are maintained by the community. The manually-curated and static nature of those lists are their main drawback, preventing them to quickly adapt to an evolving environment like web tracking.

Our proposal, like content-blockers, also works at the URL level, presenting all the associated benefits. URLs are very easy to access since it is not necessary to load the whole page or to explore any other exchanged data. Similarly, lexical features like the URL length can be measured fast, while other features such as network properties (e.g. IP Address, HTTP headers) or content properties require explicitly to download the content hosted by the URL to perform the inspection, increasing the risk and slowing down the process. Moreover, given that our method does not use code inspection, it is more robust against code minification and obfuscation, which is used by some JavaScript trackers to avoid detection [13].

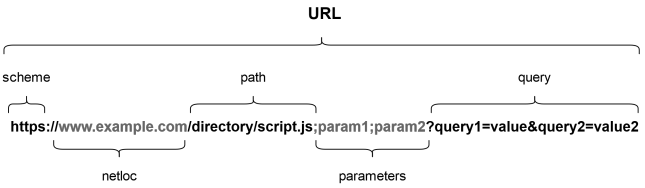

Fig. 1. URL structure

III. Proposed method

Our proposal is focused on classifying a given URL as tracking or non-tracking based only on the contents of the URL. To achieve it we have a set of URLs previously labeled, and we need to find a prediction model that minimizes the total number of mistakes in the entire data set. We are facing the problem as a binary classification problem. Considering a set of M URLs : $(x_1, y_1), (x_2, y_2), \ldots, (x_M, y_M)$ where x_m represents a URL, and $y_m \in \{0, 1\}$ is the label of the URL ($1 \rightarrow$ tracking, $0 \rightarrow$ non-tracking), we need to find a prediction function $f : R_n \rightarrow R$ whose role is to assign a class for any given instance x of type URL. The prediction made by the function is denoted as $\hat{y}_m = sign(f(x_m))$. In our case, this function is represented as a Deep Neural Network (DNN). We compared it to other models such as a Recurrent Neural Network (RNN) and a Convolutional Neural Network (CNN). Although the results were similar, RNN had much slower inference time (1.16s vs. 0.04s) and CNN had about 50% more false positives.

A. Preprocessing

The main benefit of deep learning is that it operates over raw data, making unnecessary to extract features to be used in the classification process. However, the model cannot accept directly string-format inputs like URLs. We need a preprocessing step to find a representation that could be used as input. To this end, we look at the URL as a sequence of characters, splitting it in each individual character. Every character is taken into account to give the model as much information as possible, even the separation characters (e.g. "/", ";"). At this step, each URL is a 1-dimension variable-length array such as $u = [c_1, c_2, \ldots, c_L]$ where $c_i, i = 1, \ldots, L$ is a character (e.g. letters, symbols, numbers) and L is the length of the URL.

The second step transforms each character into an integer using a dictionary preserved over all the process. Our data set contains 90 different characters numbered from 1 to 90. At the end of this step, each URL is a 1-dimension variable-length array such as $u = [n_1, n_2, \ldots, n_L]$ where $n_i, i = 1, \ldots, L$ is an integer $\in [1, 90]$ and L is the length of the URL.

The last step consist of fixing the length of the URL. Based on our expertise we selected a maximum length value of 200 characters, usually longer enough to contain all the URL (only 3.4% of the total and 6.8% of the tracking URLs are longer) but easy to handle by deep learning algorithms. The structure of a typical URL can be seen in Fig. 1. According to our observations, usually the most relevant information is contained in the *parameters* and *query* parts of the URL. As

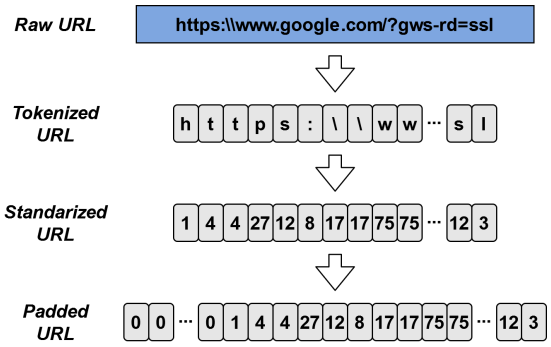

Fig. 2. URL preprocessing

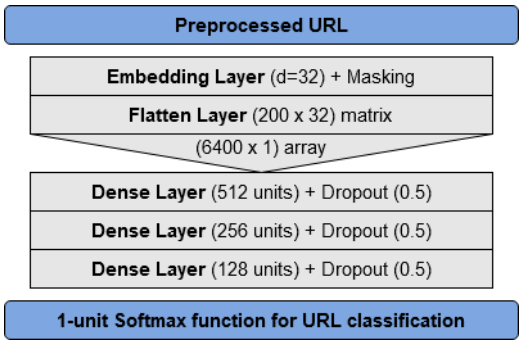

Fig. 3. Deep Neural Network architecture

shown in the figure both are located at the last positions. Consequently, if the URL is longer than 200 characters the system truncates the first elements containing the *scheme*, *domain* and *path* information until only 200 characters remain. On the contrary, if the URL is shorter than 200 characters we pad it filling it with zeros at the beginning of the array until the total length becomes 200. This way the system keeps the most important information always at the end of the array. Finishing this step, each URL is a 1-dimension fixed-length array such as $u = [0, \ldots, 0, n_1, n_2, \ldots, n_L]$ where $L = 200$ and $n_i, i = 1, \ldots, L$ is an integer $\in [1, 90]$. Fig. 2 shows a diagram of the full preprocessing process.

B. Model Architecture

Although the model proposed is prepared to receive as input the preprocessed URLs presented in the last subsection, the input still needs some manipulation to be usable by the classification process. Thus, the model itself can be conceptually divided in two parts: the URL representation generation and the classification layers. Fig. 3 shows the model complete with all its layers. The URL representation generation would comprehend the first two layers while the classification would be the last three, all of them integrated into one model.

By default deep learning treat the inputs as numerical values, interpreting that 2 close values are similar. In our scenario each character has been substituted with an integer, but there is no special relation between them. Thus, if we train the system with only the preprocessed URLs the model would be biased. To fix it we add a first *Embedding Layer*, transforming the preprocessed URLs to a low-dimensional continuous representations of them (each character will be represented by a 32-dimensions vector). Furthermore, we set a masking parameter to inform the model to ignore the zeros introduced in the padding process. Lastly, we need to simplify the resulting matrix to a 1-dimension array, the final input that *Dense Layers* expect. To this end, we add a new *Flatten Layer* that reduces the embedding output to a 1-dimension array by putting each row of the matrix side by side in one array.

The second part of the model correspond to 3 *Dense Layers* with a decreasing number of units respectively. In order to find the optimal number of layers and units per layer we executed a Grid search with multiple values. The method tries all the possible combinations and select the best one. We found that selecting too many neurons (e.g. 2048) makes the model to overfit. To avoid overfitting on the model a dropout has been added to each layer. The output is composed of a 1-unit layer with a softmax activation function giving a float output $\in [0, 1]$. The output is then rounded to the closest integer (0 for non-tracking or 1 for tracking).

TABLE I
CLASSIFICATION RESULTS

	Subset	Precision	Recall	F1 Score	Accuracy
DNN + Left Padding	Non-tracking	95%	100%	0.97	94.7%
	Tracking	99%	54%	0.69	
DNN + Left Padding + Embedding	Non-tracking	98%	100%	0.99	97.1%
	Tracking	97%	78%	0.86	
DNN + Right Padding + Embedding	Non-tracking	97%	100%	0.98	97.9%
	Tracking	96%	83%	0.87	

IV. Evaluation and Visualization

A. Evaluation

To evaluate the system we analyzed the top 100K most popular websites as per Alexa's list [14]. The exploration was performed on April 2020 using ORM [2], an open source framework we developed to map URL and resources used within online domains. The labeling of the ground-truth was done using *uBlock Origin* [20] running over a dozen filter-lists including EasyList [21] and EasyPrivacy [22], two of the most popular and precise pattern lists available. The collected data set has more than 5,3M unique URLs where approximately 4,7M have been labeled as non-tracking and 600K as tracking. The evaluation of DTD was made using cross-validation (90% for training and 10% for testing) over the entire data set. Furthermore, we used a subset of the training set to stop the training process when overfitting starts.

Table I shows the results of DPD with different configurations for the embeddings and tagging process. The results are splitted differentiating non-tracking and tracking classification. The system has high accuracy with all the configurations, presenting the best results by using an embedding and padding/truncating the URLs by the right side. However, as seen in Fig. 1 and commented before, padding or truncating them by the right would require to truncate requests *parameters* and *query* properties, for long URLs. These fields include relevant information, such as the file type (e.g. most tracking systems work on JavaScript files). On the contrary,

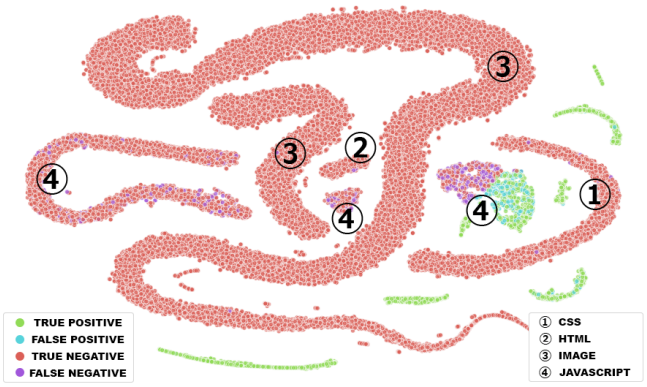

Fig. 4. t-SNE applied on the output of the penultimate layer of the model

the left fields include information about the *domain* names. Thus, padding the URLs by the right could bias the system towards finding common properties of the *domains* hosting tracking, instead of focusing on the properties of the resources themselves. Accordingly, we decided our system to pad the URLs by the left, even if the accuracy is a bit lower with our data set ($\approx$ 0.8%). This way, the resulting model should be more robust and suffer less from temporal obsolescence, for example if the list of tracking domains changes over time.

For the selected configuration (DNN + Left Padding + Embedding) the system present a very high accuracy specially for the non-tracking classification. For tracking URLs the precision is very high with a 97%, but the recall decreases until 78%. Blocking non-tracking resources may break website usability or functionality. Thus, from the user perspective it is better to have more false negatives and a bit lower recall, letting pass through some tracking URLs, than having a lower precision and higher false positives.

B. Visualization and Observations

In order to have a better understanding of the resulting model, we applied some visualization techniques using the output of the penultimate layer of the model. We collected each element, represented by a 128-dimension array, and applied a dimensionality reduction technique called t-SNE. The resulting plot, shown in Fig. 4, presents the classification accuracy in different colors. There are clear clusters for tracking as well as non-tracking URLs. It also presents one cluster with mixed results. We correlated some characteristics of the URLs included in each cluster to discover particularities and, as expected, we found differentiated clusters depending on the type of file being loaded by the URL. This enforces our decision of padding the URLs by the left to maintain file characteristics. The numbers of the figure represent the most common resource for each cluster. The small green clusters are all formed by JavaScript files.

To explore the mixed cluster in higher detail, we randomly selected a subset of 50K URLs and loaded the information with the online visualization tool from Tensorflow [23] using the same t-SNE technique. The tool permitted us to explore

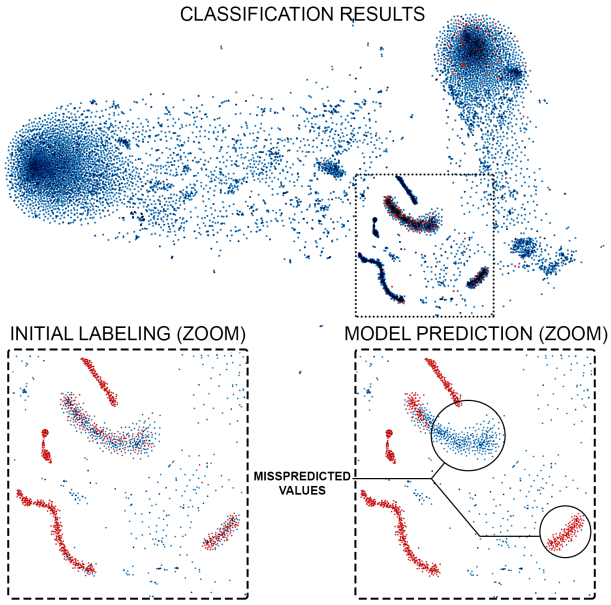

Fig. 5. Classification results detail

the details of each URL. Fig. 5 shows the results given by the visualization tool, differentiating between correctly classified in blue and badly classified in red. The clusters in the center of the figure contain the majority of errors. We zoomed in those clusters and compared the initial labels given by the pattern lists with the predicted values (red→tracking, blue→non-tracking). Most of the incorrectly classified URLs belong to the clusters inside the circles shown in the figure. Inspecting the corresponding URLs we found that most errors pertain to ground-truth misclassified URLs for very common non-tracking libraries like *JQuery*, or *WordPress* configuration scripts. This explains why our method also misclassified them and highlights the inaccuracies of current pattern lists.

V. Conclusions

In this work we presented Deep Tracking Detector (DTD), a new web tracking detection system based on deep learning. DTD can detect web tracking resources with 97% accuracy using only as input the characters of the URL string. To the best of our knowledge this is the first work to apply deep learning to the detection of web tracking using only the URL. Our solution is simple and much easier to adopt than more complex systems recently proposed in the literature. Compared to current content-blockers, our solution is more difficult to evade and can potentially generalize to new tracking domains that may appear in the future, if they have similar characteristics to current tracking domains. As a future work we plan to develop browser plugins that implement our model for the most popular browsers, which could help Internet users to easily improve their privacy.

VI. Acknowledgments

This work was supported by the Spanish MINECO under contract TEC2017-90034-C2-1-R (ALLIANCE).

REFERENCES

[1] S. Englehardt and A. Narayanan, "Online Tracking: A 1-million-site Measurement and Analysis," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, (Vienna, Austria), pp. 1388–1401, Association for Computing Machinery, Oct. 2016.

[2] I. Castell-Uroz, J. Solé-Pareta, and P. Barlet-Ros, "Network measurements for web tracking analysis and detection: A tutorial," *IEEE Instrumentation & Measurement Magazine*, vol. Special Issue "Measurements for Advanced Networking & Networks for Advanced Measurements", In press.

[3] J. Mikians, L. Gyarmati, V. Erramilli, and N. Laoutaris, "Crowd-assisted search for price discrimination in e-commerce: first results," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, CoNEXT '13, (Santa Barbara, California, USA), pp. 1–6, Association for Computing Machinery, Dec. 2013.

[4] K. Lobosco, "Facebook friends could change your credit score," Aug. 2013. https://money.cnn.com/2013/08/26/technology/social/facebook-credit-score/index.html.

[5] The Economist, "Very personal finance." https://www.economist.com/finance-and-economics/2012/06/02/very-personal-finance.

[6] "Web Analytics Market Size, Share and Industry Analysis | Forecast 2026." https://www.alliedmarketresearch.com/web-analytics-market-A05971.

[7] U. Iqbal, P. Snyder, S. Zhu, B. Livshits, Z. Qian, and Z. Shafiq, "ADGRAPH: A Graph-Based Approach to Ad and Tracker Blocking," *IEEE Symposium on Security and Privacy 2020*, p. 14, 2019.

[8] Q. Wu, Q. Liu, Y. Zhang, P. Liu, and G. Wen, "A Machine Learning Approach for Detecting Third-Party Trackers on the Web," in *Computer Security – ESORICS 2016* (I. Askoxylakis, S. Ioannidis, S. Katsikas, and C. Meadows, eds.), Lecture Notes in Computer Science, (Cham), pp. 238–258, Springer International Publishing, 2016.

[9] M. Ikram, H. J. Asghar, M. A. Kaafar, A. Mahanti, and B. Krishnamurthy, "Towards Seamless Tracking-Free Web: Improved Detection of Trackers via One-class Learning," *Proceedings on Privacy Enhancing Technologies*, vol. 2017, pp. 79–99, Jan. 2017.

[10] J. Mazel, R. Garnier, and K. Fukuda, "A comparison of web privacy protection techniques," *Computer Communications*, vol. 144, pp. 162–174, Aug. 2019.

[11] V. Kalavri, J. Blackburn, M. Varvello, and K. Papagiannaki, "Like a Pack of Wolves: Community Structure of Web Trackers," in *Passive and Active Measurement* (T. Karagiannis and X. Dimitropoulos, eds.), Lecture Notes in Computer Science, (Cham), pp. 42–54, Springer International Publishing, 2016.

[12] H. Metwalley, S. Traverso, and M. Mellia, "Unsupervised Detection of Web Trackers," in *2015 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, Dec. 2015.

[13] H. Le, F. Fallace, and P. Barlet-Ros, "Towards accurate detection of obfuscated web tracking," in *2017 IEEE International Workshop on Measurement and Networking (M N)*, pp. 1–6, Sept. 2017.

[14] K. Cooper, "Alexa: Most popular website list," 2020. https://www.alexa.com/.

[15] T. Bujlow, V. Carela-Español, J. Solé-Pareta, and P. Barlet-Ros, "A Survey on Web Tracking: Mechanisms, Implications, and Defenses," *Proceedings of the IEEE*, vol. 105, pp. 1476–1510, Aug. 2017.

[16] Z. Yu, S. Macbeth, K. Modi, and J. M. Pujol, "Tracking the Trackers," in *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, (Montréal, Québec, Canada), pp. 121–132, International World Wide Web Conferences Steering Committee, Apr. 2016.

[17] H. Le, Q. Pham, D. Sahoo, and S. C. H. Hoi, "URLNet: Learning a URL Representation with Deep Learning for Malicious URL Detection," *arXiv:1802.03162 [cs]*, Mar. 2018.

[18] AdBlock Plus, "Adblock Plus," 2020. https://adblockplus.org/en/.

[19] Ghostery, "Ghostery Makes the Web Cleaner, Faster and Safer!," 2020. https://www.ghostery.com/.

[20] R. Hill, "uBlock Origin," 2020. https://github.com/gorhill/uBlock.

[21] "EasyList," Feb. 2020. https://easylist.to/.

[22] "EasyPrivacy," 2020. https://easylist.to/easylist/easyprivacy.txt.

[23] N. Thorat, C. Nicholson, Big, and D. Smilkov, "Embedding projector - visualization of high-dimensional data." http://projector.tensorflow.org.