A NOVEL APPROACH TO USER AGENT STRING PARSING FOR VULNERABILITY ANALYSIS USING MULTI-HEADED ATTENTION

DHRUV NANDAKUMAR, SATHVIK MURLI, ANKUR KHOSLA, KEVIN CHOI,
ABDUL RAHMAN, DREW WALSH, SCOTT RIEDE, ERIC DULL, EDWARD BOWEN

Deloitte & Touche LLP
E-MAIL: kevchoi@deloitte.com

Abstract:

 The increasing reliance on the internet has led to the proliferation of a diverse set of web-browsers and operating systems (OSs) capable of browsing the web. User agent strings (UASs) are a component of web browsing that are transmitted with every Hypertext Transfer Protocol (HTTP) request. They contain information about the client device and software, which is used by web servers for various purposes such as content negotiation and security. However, due to the proliferation of various browsers and devices, parsing UASs is a non-trivial task due to a lack of standardization of UAS formats. Current rules-based approaches are often brittle and can fail when encountering such non-standard formats. In this work, a novel methodology for parsing UASs using Multi-Headed Attention Based transformers is proposed. The proposed methodology exhibits strong performance in parsing a variety of UASs with differing formats. Furthermore, a framework to utilize parsed UASs to estimate the vulnerability scores for large sections of publicly visible IT networks or regions is also discussed. The methodology present here can also be easily extended or deployed for real-time parsing of logs in enterprise settings.

Keywords:

 User Agent String; Natural Language Processing; Parsing; Transformer

1. Introduction

The increasing reliance on the internet has led to the proliferation of a diverse set of web-browsers and operating systems (OSs) capable of browsing the web. Each of these software packages is versioned separately and has separate security vulnerabilities as an artifact of their design. The vulnerabilities have the potential to be exploited by malicious actors as a method for data theft, network intrusion, or a variety of other tasks.

User agent strings (UASs) are a component of web browsing that are transmitted with every Hypertext Transfer Protocol (HTTP) request. They contain information about the client device and software, which is used by web servers for various purposes such as content negotiation and security. UASs contain information pertaining the names and versions of OS, web browser, hardware, and software components used by a client device. This information can be used to assess the vulnerabilities of devices by correlating the extracted information with known Common Vulnerabilities and Exposures (CVEs). These vulnerabilities can then be quantified using the Common Vulnerability Scoring System (CVSS) scores which represent the severity of a vulnerability [1]. If done at a larger scale, CVSS scores can be aggregated at regional levels to estimate the vulnerability and security levels of large portions of the public internet.

However, the format of UASs is not standardized and can vary greatly, making it difficult to parse and interpret their contents accurately. This can have significant implications for web security, as incorrect information can lead to incorrect decisions being made by web servers, such as serving content that is not compatible with the client device, or failing to properly secure the connection. Furthermore, inaccurate parsing of UASs could significantly impact the estimation of CVSS scores at the device level as CVEs and CVSS scores are heavily reliant on the names and versions of browsers and OSs.

Conventional methods for parsing UASs typically rely on rules-based approaches, where regular expressions or heuristics are used to extract relevant security information. While these methods can be effective for well-formed UASs, they are often brittle and can fail when encountering strings with unexpected or non-standard formats. This can lead to incorrect information being extracted or in many cases, not extracted at all.

In this paper, we propose a novel approach for parsing UASs

using transformer-based natural language processing (NLP) techniques. Our method is designed to be robust and capable of handling a wide range of UAS formats, while accurately extracting information about the client device and software. Furthermore, we also introduce a methodology for estimating the vulnerability of large sections of public internet by correlating UAS information to CVEs.

The key contributions of this paper are two-fold:

- A robust method for parsing UASs using transformer-based NLP techniques, which overcomes the limitations of traditional rules-based methods.

- The use of extracted fields from the parsing process to assess the vulnerabilities of large portions of the internet by correlating the extracted information with known CVEs, which can help to identify potential security risks and inform mitigation strategies.

The accuracy and reliability of UAS parsing is critical for ensuring the security and stability of the internet. By leveraging the information contained in UASs to identify vulnerabilities in networks, our approach has the potential to significantly improve the current state and provide valuable insights into the security of the internet.

2. Related Work

UASs have been used extensively to provide content distribution and web servers with the information required for said servers to provide optimal web content. Since the advent of UASs, there has been significant progress in their standardization from a browser perspective. Several browsers and organizations have defined protocols and issued guidance on standardizing UAS formats including measures to shorten UASs and omit non-essential or personally identifiable fields. However, there does not yet exist a universally standardized UAS format. There are several cases when a browser cookie adds its own information in the UAS, or UASs originating from in-app browsers consisting of multiple software names. These cases introduce significant variability to UAS formats which weaken rules-based approaches to UAS parsing; thus, justifying the need for the development of more robust methods.

There has been work completed on data relating UASs in similar fashion to the solution proposed in this paper. Crucially however, the scopes of work, methodologies used, and resulting outputs differ from ours. For example, work has been completed using pre-processing methods related to NLP in the past. Zhange et al. [2] used context-free grammars to distinguish fake UASs from real ones. Want et al. [3] converted HTTP flow headers into N-gram sequences to obtain a bag-of-words representation. This resulting representation was then fed into a support vector machine to identify malicious network traffic. Tanaka et al. [4] implemented a similar bag-of-words representation, but they used logistic regression, and a tree-based model called Light Gradient-Boosting Machine (LightGBM) [5].

UASs have been core source of information where the aim is to detect malicious activities using user logs. The notion of all these works is to extract information from the UAS, encode this information using rule-based techniques and perform statistical analysis on the generated features. For example, Chen et al. [6] used regex on UASs to detect anomalous user agents in network traffic. Boda et al. [7] described a structure that the majority of UASs they collected follow and leveraged that information for browser fingerprinting. Grill et al. [8] divided UASs based on different types, for example legitimate UA, spoofed, empty, etc. Statistical analysis was performed post classification. Lewis et al. [9] also uses rule-based approach for parsing, taking into account the defined HTTP structure of UAS. Rule-based approaches have been quite prevalent in this domain.

However, multiple works have used NLP-related models on entire HTTP requests to discriminate between malicious and benign traffic. Attempts have been made to perform character level encoding (e.g., 0 for vocabulary characters, 1 for numeric values, etc.) to identify patterns and correlation among multiple UASs. Gao et al. [10] used this approach along with status code, content length and referrer as feature vectors. Post feature extraction, clustering algorithms were used to identify malicious users. Zolotukhin et al. [11] used an N-gram model to extract features from UAS, followed by mathematical modeling and Principal Component Analysis (PCA) [12] combined with Support Vector Data Distribution (SVDD) [13], K-means [14], Density-based spatial clustering of applications with noise (DBSCAN) [15] and aggregated time bin for analysis. Rong et al. [16] used a character-level convolutional neural network, which is a convolutional neural network operating on character-level embeddings of HTTP requests. Park et al. [17] used a similar method, feeding character-level embeddings into an auto-encoder. Gniewkowski et al. [18] leveraged byte-pair tokenization to generate inputs that were then fed into the transformer-based model, a Robustly Optimized BERT Pre-training Approach (RoBERTa) [19]. A key characteristic of these works is they do not extract individual pieces of information from the UASs, but process them as part of a larger feature set.

The collective nature of UAS analysis has resulted in reduced focus towards feature extraction from independent UASs. The majority of prior work revolved around identifying patterns

from a collection of UASs / web logs. However, in this paper we focus on UAS at rudimentary level to extract features and present threat correlation as an extended use case of this approach. The objectives of our work are to present a foundational approach to UAS parsing and usage in enterprise security and IT settings.

3. Problem Setup

In order to build a more robust UAS parser, we propose the use of NLP techniques including Multi-Headed Attention. The scope of our modeling efforts are limited to parsing 4 specific pieces of information from a UAS, although we believe that this methodology can be extended to most other parts of a UAS. Particularly, we focus on extracting:

- OS Name and Version

- Browser (Software) Name and Version

In the following subsections, we explore the structure of our training and validation data, our model architectures, and experimental setup.

3.1. Data and Pre-processing

Our training dataset consists of over 200 million public UASs collected by WhatIsMyBrowser.com [20]. Each UAS in the dataset was labelled with Software and OS names and versions, along with several other pieces of UAS metadata. We treat this dataset as a labelled dataset with ground truth being the labels assigned by the issuing entity.

We pre-processed each UAS to remove special characters, parentheses, remnants of HTML and to standardize whitepsace by performing the substitutions referenced in Table 1. Furthermore, we also limited the length of a UAS to 50 words after preprocessing, wherein any UAS longer than 50 words would be truncated at length 50.

From the pre-processed data, a balanced training dataset was curated for software name and OS name classification. There were a total of 7 classes in each of the Software and OS names. (Software names – Android WebView, Chrome, Facebook App, Internet Explorer, Instagram, Opera, and N/A. OS names – Android, iPad, iOS, Linux, Macintosh, Windows, and N/A). These (top 6) classes were selected based on their popularity among 200M examples. 'N/A' class denotes that the UAS belonged to none of the above six classes. For software name classification and version identification models, each class had 1.4M randomly sampled examples, making the total size of training

TABLE 1. Character edits made to format the UASs.

Original Character	Replacement
"%20"	" "
"_"	"."
"\("	"("
"\)"	")"
"/"	" "
";"	Removed
":"	" : "
"%"	" "

data almost 9.8M. For OS name classification and version identification models each class had 2M examples and 500k for N/A class, making the total training dataset size as 12.5M.

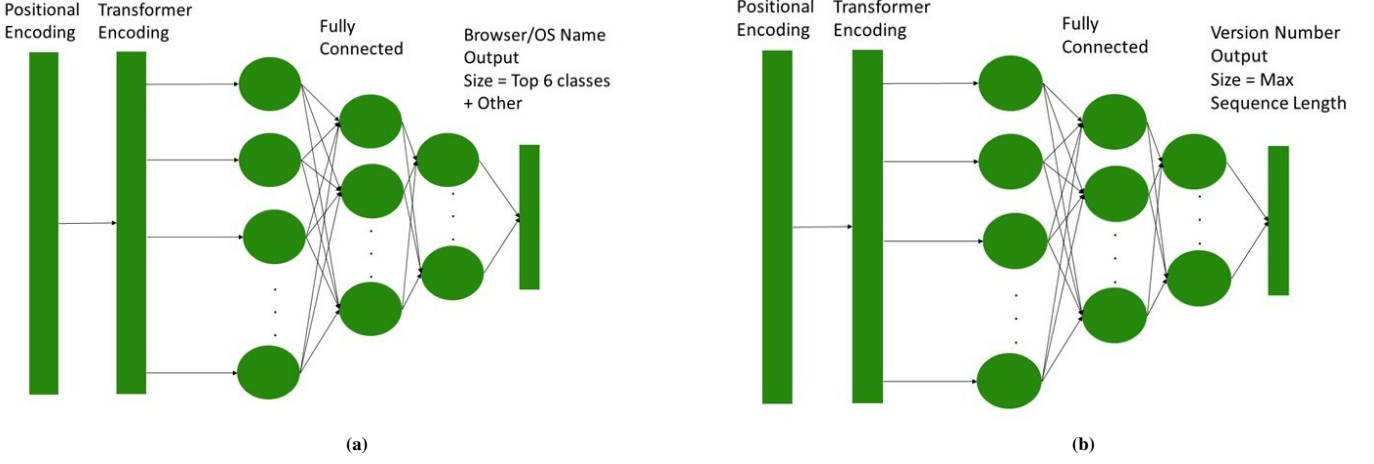### 3.2. Modeling Considerations and Architectures

In order to parse OS and Software names and versions from UASs, we leverage 4 independent models - one for each task.

Parsing of OS and Software version numbers is treated as a part of speech (PoS) tagging problem wherein two independent models are trained to output a likelihood of each word in a UAS corresponding to the versions of the OS and Software respectively. Conversely, due to the fact the OS and Software names are often not explicitly stated in UASs, we treat their parsing as a classification problem where two models are trained to classify the names of the target fields using the entire UAS as context respectively.

Each of the four models consist of the following three components:

3.2.1. UAS Embeddings

Given that the content of UASs are domain specific, we cannot use pre-existing tokenizers and embedding models to produce vector embeddings of UASs. Furthermore, we also want to leverage information about character substrings in words of a UAS. For example, we want to capture the similarity between the strings *Mac* and *Macintosh* rather than treat them as separate words and tokens. Consequently, we utilize Fasttext [21] embeddings trained on a corpus of UASs to represent each word in a UAS as a vector. The Fasttext model is trained using a Continuous Bag Of Words (CBOW) approach and produces a 1-dimensional vector of length 40 per word in a UAS, where a word is any character sequences separated by a space after

FIGURE 1. Model architectures for (a) classification of software name and OS name (b) predicting location of software version and OS version

preprocessing a UAS. Consequently, each UAS will be represented by a vector with length 50 and width 40 after passing through the embedding layer.

3.2.2. Representation Layer

Each of the four models share the same architecture at this layer. Embeddings from the UAS Embedding layer are passed through a positional encoding layer followed by a single transformer encoder layer with two attention heads. A positional encoding layer is used in attention-based neural networks to convey the positional information that would not normally be captured by a transformer-based network [22]. Our positional encoding layer is defined by the following function:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}}) \qquad (1)$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}}) \qquad (2)$$

In which d_{model} refers to the length of the generated word embeddings and i refers to the row upon which the operation is being performed. pos refers to the position within each row or sequence that is being passed through the layer. The output of this layer is flattened into a one dimensional vector of length *2000* (from flattening a *50* x *40* vector).

3.2.3. Task Specific Heads

Both the OS and Software name classification heads consist of fully connected dense layers. They are terminated by layers of seven nodes, six of which are for specific classes and the

seventh signifying an 'N/A' class. The output of the terminal layers are passed through a softmax function. The PoS models for version identification also feature fully connected layers but are terminated by a layer with 51 nodes; one for each word in the UAS and another indicating the version was not present in the UAS. The fully connected layers are separated by SeLU activation functions. During the training process, there are also dropout layers that follow the first 3 fully connected layers to combat the risk of overfitting. The overall architecture for each of the models is represented in Figure 1.

3.3. Experimental Setup

Throughout the experiments in this paper, we utilized a 70%/30% training and validation split of the data where classes were sampled using Random Stratified Sampling to have appropriate representation in both training and validation data. We also utilized a fixed batch size of 200 UASs per batch for each of the experiments in this work.

3.3.1. Version Indexing models

The optimizer used in the case of Version Indexing models was Stochastic Gradiatent Descent (SGD) with a learning rate of 0.005 and weight decay of 0.00001. The model was trained using a Cross Entropy loss function. The word at the index with the highest raw value was selected as the version.

3.3.2. Name Classification models

Here, we utilized the SGD optimizer with a learning rate of 0.0005 and weight decay of 0.00001 and a Binary Cross Entropy loss function. A softmax activation function was used in the last layer to compute the probability scores of each class. Class with highest probability was selected as the final output.

3.4. Post Processing

After extraction of the relevant fields from a UAS, our objective is to perform vulnerability analysis on the Classless Inter-Domain Routing (CIDR) ranges from which the UASs originate. We monitor the UASs originating from a particular CIDR range. Once we extract system information (Software and OS) from the UAS, we then estimate a vulnerability score for that particular UAS. For this purpose, we use National Vulnerability Database (NVD: https://nvd.nist.gov/). NVD is a database, maintained by U.S. government, where Subject Matter Resources (SMRs) analyze the vulnerabilities, based on predefined metrics and score them on a scale of 1-10 (10 being highly vulnerable). Although there are mix of old and new scoring systems, such as v2.0 and v3.1, we consider the latest available scores for each vulnerability.

These scores can be computed using CVSS. Upon analyzing several factors, such as attack vectors and attack complexity, CVSS computes a score based on base, temporal and environmental metrics (https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator). There are three types of scores provided for each vulnerability – Base Score, Exploitability Score and Impact Score.

To map the system information with existing vulnerabilities, we use Common Platform Enumeration (CPE) names (https://nvd.nist.gov/products/cpe). For a single UAS, each of the CPE names and their vulnerability scores are computed using the four-tuple *(OS Name, OS Version, Software Name, Software Version)*. To estimate the vulnerability of a UAS, we compute the mean CVSS score for that UAS accross all known CVEs. This process is conducted for an average Base Score, Impact Score, and Exploitability Score. The averages are given by the equations:

$$Vul_{(base\_uas)} = \frac{1}{n}\sum_{i=i}^{n} CVSS_{base_i} \qquad (3)$$

$$Vul_{(exploit\_uas)} = \frac{1}{n}\sum_{i=i}^{n} CVSS_{exploit_i} \qquad (4)$$

$$Vul_{(impact\_uas)} = \frac{1}{n}\sum_{i=i}^{n} CVSS_{impact_i} \qquad (5)$$

Where

$$CVSS_X i$$

represents the CVSS score for the Base, Exploitability, or Impact score for the *ith* CPE and *n* is the total number of CPEs identified for a UAS.

Once we have the vulnerability scores corresponding to each UAS, we map these scores back to the CIDR ranges. This score distribution is analyzed for each CIDR range to estimate the exposure of the endpoints / systems in that network. We can visualize this vulnerability score distribution against the geographical locations to gauge the exposure on the map. Another way is to analyze this distribution dynamically and visualize how the exposure of a particular CIDR range changes over time. The workflow for the aforementioned vulnerability estimation process is presented in Figure 2. NVD Application Programming Interfaces (APIs) were used for CVE Vulnerability Listing and CVSS Vulnerability Scoring.

4. Results

Our approach has shown strong results on the 4 tasks we use for evaluation. For the tasks structured as classification problems, accuracy, precision, recall and f1-scores were generated. For the tasks involving location of version numbers, accuracy is the measure of choice.

4.1. Name Classification Results

Tables 3 and 4 show the accuracy, precision, and recall of our approach on classification of both OS names and software names. The support column describes the number of rows within the validation set that correspond to each respective class. The results show the model performance is very high on previously unseen data, with the models able to accurately classify strings based on which OS name or browser name they have. Further evaluation runs of the models show their ability to detect strings in which the OS name may be in varying positions or represented as abbreviations or acronyms. Examples of these types of situations are displayed in Table 2. A yellow highlight indicates a false indicator, a green highlight indicates the correct answer. The correct answers our model classifies are Android and Facebook respectively. The results also show performance does suffer on strings corresponding to Linux OSs, due to its overlap with certain other less frequently
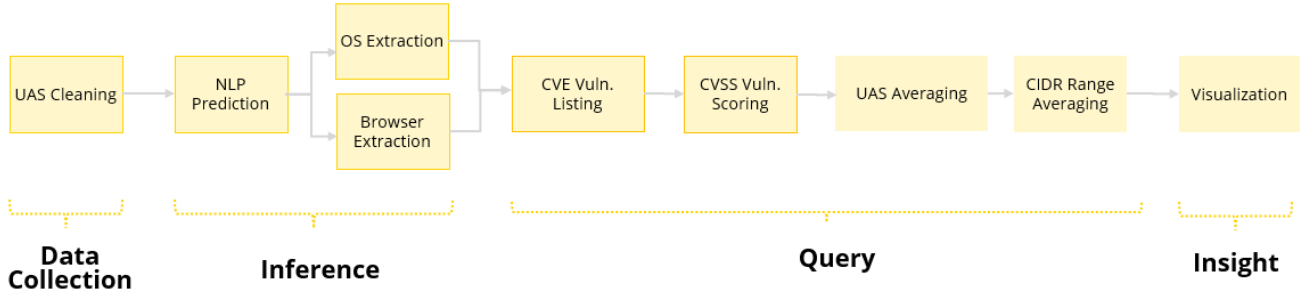
FIGURE 2. Vulnerability Estimation Workflow

TABLE 2. Examples of our model's ability to differentiate between false indicators and the correct answer. A yellow highlight indicates a false indicator, a green highlight indicates the correct answer. The correct answers our model detects are Facebook and Android, respectively.

UAS	False Indicator (Shown in Yellow)	Correct Indicator (Shown in Green)	Generated Answer
Mozilla 5.0 (Linux Android 12 M2101K9AG Build SKQ1.210908.001 wv) AppleWebKit 537.36 (KHTML, like Gecko) Version 4.0 Chrome 104.0.5112.97 Mobile Safari 537.36 5bFB.IAB Orca-Android FBAV 377.0.0.13.101 5d	Chrome	5bFB.IAB, FBAV	Facebook
Mozilla 5.0 (Linux Andr0id 10 BRAVIA 4K VH2) AppleWebKit 537.36 (KHTML, like Gecko) Chrome 84.0.4147.125 Safari 537.36 OPR 46.0.2207.0 OMI 4.21.0.273.DIA6.142	Linux	Andr0id	Android

encountered OSs that were also UNIX based. Android would be an example of such an OS, but Android OSs were strongly represented in the training data. The results show no such issues with software name classification, which has consistent performance across the 6 most frequently encountered classes.

TABLE 3. Classification of Software Names

Class Name	Precision	Recall	F1 Score	Support
Facebook	1.00	1.00	1.00	285551
Instagram	1.00	1.00	1.00	286175
Chrome	0.98	0.98	0.98	285615
Android Webview	0.97	0.99	0.98	283844
Internet Explorer	0.99	1.00	1.00	285320
Opera	1.00	1.00	1.00	282903
N/A	0.98	0.95	0.97	284192

TABLE 4. Classification of OS Names

Class Name	Precision	Recall	F1 Score	Support
Android	1.00	1.00	1.00	7038987
iOS	0.99	1.00	1.00	1281009
Windows	0.98	0.98	0.98	1006057
Macintosh	0.96	0.90	0.93	115541
Linux	0.99	0.87	0.92	116444
iPad	0.98	0.95	0.97	181760
N/A	0.87	0.91	0.89	121438

4.2 Version Indexing Results

For evaluation of version number identification models, accuracy is our primary measure. Table 5 shows the model performance on both the version number identification tasks. The models have strong accuracy for detecting both kinds of version numbers. Table 6 displays certain OS version numbers that had

TABLE 5. Version Number Indexing Results

Task	Overall Accuracy
Software Version Number	99.4%
OS Version Number	99.5%

TABLE 6. Version Number Indexing Examples

Version Number	Precision	Recall	F1 Score	Support
2022060972	0.996	0.99	0.993	1732
21.113	0.948	0.998	0.973	2341
12.4.6	0.998	1.000	0.999	3215
10.0	1.000	1.000	1.000	424575

FIGURE 3. Vulnerability Assessment (on Base Scores) vs CIDR ranges - Geographical analysis.

lower occurrences in the testing set but were still detected by the model. The final row in the table shows the version number with the most occurrences in the testing set for reference. These results display the potential for the models to identify version numbers in varying scenarios and formats. Thus the model shows very strong coverage for different software types and OSs.

4.3. Vulnerability Visualization

Table 7 shows the Vulnerability Scoring system for a few examples in our data. Six UASs are mentioned for two CIDR ranges. Software name and OS information is extracted from these UASs, and a base vulnerability score is computed for each UAS accordingly. The *Avg. Base Score* column represents average of all the base scores corresponding to a particular CIDR range. Furthermore, the CIDR range to base score mapping can be extended to identification of vulnerable systems in a network.

Figure 3 depicts the visualization built on the top of CIDR ranges against Base Vulnerability Scores after post-processing. This visualization is based only on a subset of our data. However, we believe that the illustration is representative of the promise our proposed approach holds for vulnerability monitoring and trend analysis on a per-network or region level. These visualizations could help security experts analyse changing trends to the security posture of their internal networks or networks to which they connect. It can also assist in the cases where SMRs are attempting to patch vulnerable endpoints in a network.

5. Conclusion and Future Work

In this work, we have introduced a novel, foundational approach to UAS parsing that is scalable, extensible, and most importantly, robust to the varying formats of UASs. With strong performance in several categories, we demonstrate that our proposed approach also shows promise in enterprise settings where reliable parsing of UASs is of paramount importance.

We have also demonstrated how our approach could ostensibly be used in cybersecurity-specific contexts for vulnerability trends analysis on a network or regional level. While not in the scope of this work, we believe that our methodology can be adapted and utilized in several other fields of application where the independent data consist of strings that have a complex, partially-standardized structure and cannot be tokenized using tokenizers for common natural languages.

As for future work, we believe that there can be several avenues of research for improving the performance of our models. The primary examples of one such direction is improving the performance of our parsers on very-low-support classes, or few-shot learning, for parsing rare UASs such as UASs from smart speakers. We are also looking to improve the performance on models on difficult examples, as seen with the *Linux* OS in our experimental results. We have seen promising initial results along this line of inquiry by utilizing curriculum learning to train our models to parse easier examples first before introducing them to harder examples. Our approach uses a weighted average of the Euclidean distance of a chosen UAS's word embedding from the mean word embeddings, the difference in length of the string from the mean length, and the output of the name classification models. Given the initially strong experimental results, we believe that this research direction also warrants further investigation.

TABLE 7. Scoring system examples on CIDR Ranges

CIDR Range	User Agent Strings	Software	Operating System	Base Score (from CPE names)	Avg. Base Score
1.123.**.*/24	Mozilla 5.0 (Windows Phone 8.1 ARM Trident 8.0 Touch rv : 11.0 IEMobile 11.0)	Internet Explorer, 11	Windows 8.1	6.15814104	5.977965983
	Mozilla 5.0 (Linux Android 12 SM-G986B) AppleWebKit 537.36 (KHTML, like Gecko) Chrome 105.0.0.0 Mobile	Chrome, 105.0.0.0	Android, 12	6.57974525	
	Mozilla 5.0 (Linux Android 11 SM-G988B) AppleWebKit 537.36 (KHTML, like Gecko) Chrome 105.0.0.0 Mobile	Chrome, 105.0.0.0	Android, 11	6.07651594	
	......	......	......	......	
	......	......	......	......	
101.127.**.*/24	Mozilla 5.0 (Linux Android 9 SM-N960F Build PPR1.180610.011 wv) AppleWebKit 537.36 (KHTML, like Gecko) Version 4.0 Chrome 105.0.5195.136 Mobile Safari 537.36 SheinApp(shein 8.5.6) TTID hybrid@wing.android.1.0.1	Android WebView, 105.0.5195.136	Android	5.79486228	5.947666269
	Mozilla 5.0 (Linux Android 12 SM-A515F Build SP1A.210812.016 wv) AppleWebKit 537.36 (KHTML, like Gecko) Version 4.0 Chrome 105.0.5195.136 Mobile Safari 537.36 5bFB.IAB Orca-AndroidFBAV 379.1.0.23.114 5d	Facebook App, 379.1.0.23.114	Android, 12	5.80187262	
	Mozilla 5.0 (Windows NT 10.0 WOW64 Trident 7.0 NMTE rv : 11.0)......	Internet Explorer, 11	Windows, 10	6.39906788	
	......	......	......	......	
	......	......	......	......	

References

[1] T. M. Corporation. [Online]. Available: https://cve.mitre.org/.

[2] Y. Zhang, H. Mekky, Z.-L. Zhang, R. Torres, S.-J. Lee, A. Tongaonkar, and M. Mellia, "Detecting malicious activities with user-agent-based profiles," pp. 3016-316, 2015

[3] S. Wang, Q. Yan, Z. Chen, B. Yang, C. Zhao, and M. Conti, "Detecting android malware leveraging text semantics of network flows," IEEE Transactions on Information Forensics and Security, vol. 13, no. 5, pp. 1096–1109, 2017.

[4] T. Tanaka, H. Niibori, S. Li, S. Nomura, H. Kawashima, and K. Tsuda, "Bot detection model using user agent and user behavior for web log analysis," Procedia Computer Science, vol. 176, pp. 1621–1625, 2020, Knowledge-Based and Intelligent Information Engineering Systems: Proceedings of the 24th International Conference KES2020.

[5] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," Advances in neural information processing systems, vol. 30, 2017.

[6] J. Chen, G. Gou, and G. Xiong, "An analysis of anomalous user agent strings in network traffic," in 2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems(HPCC/SmartCity/DSS), pp. 1771–1778, 2019.

[7] K. Boda, Á M. Földes, G. G. Gulyás, and S. Imre, "User tracking on the web via cross-browser fingerprinting," in Information Security Technology for Applications: 16th Nordic Conference on Secure IT Systems, NordSec 2011, Tallinn, Estonia, October 26-28, 2011, Revised Selected Papers 16. Springer, pp. 31–46, 2012.

[8] M. Grill and M. Rehák, "Malware detection using http user- agent discrepancy identification," in 2014 IEEE International Workshop on Information Forensics and Security (WIFS). IEEE, pp. 221–226, 2014.

[9] T. Lewis, "Http header heuristics for malware detection," SANS Institute InfoSec Reading Room, 2013.

[10] Y. Gao, Y. Ma, and D. Li, "Anomaly detection of malicious users' behaviors for web applications based on web logs," in 2017 IEEE 17th International Conference

on Communication Technology (ICCT), pp. 1352–1355, 2017.

[11] M. Zolotukhin, T. Hämäläinen, T. Kokkonen, and J. Siltanen, "Analysis of http requests for anomaly detection of web attacks," in 2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing, pp. 406–411, 2014.

[12] I. T. Jolliffe, Principal component analysis for special types of data. Springer, 2002.

[13] D. M. Tax and R. P. Duin, "Support vector data description," Machine-learning, vol. 54, pp. 45–66, 2004.

[14] M. Ahmed, R. Seraj, and S. M. S. Islam, "The k-means algorithm: A comprehensive survey and performance evaluation," Electronics, vol. 9, no. 8, pp. 1295, 2020

[15] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, ser. KDD'96. AAAI Press, pp. 226–231, 1996.

[16] W. Rong, B. Zhang, and X. Lv, "Malicious web request detection using character-level cnn," in Machine Learning for Cyber Security: Second International Conference, ML4CS 2019, Xi'an, China, September 19-21, 2019, Proceedings 2. Springer, pp. 6–16, 2019

[17] S. Park, M. Kim, and S. Lee, "Anomaly detection for http using convolutional autoencoders," IEEE Access, vol. 6, pp. 70884–70901, 2018.

[18] M. Gniewkowski, H. Maciejewski, T. R. Surmacz, and W. Walentynowicz, "Http2vec: Embedding of http requests for detection of anomalous traffic," arXiv preprint arXiv:2108.01763, 2021.

[19] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," arXiv preprint arXiv:1907.11692, 2019.

[20] WhatIsMyBrowser, "Browse our database of 219.4 million user agents," 2023.[Online]. Available: https://explore.whatismybrowser.com/useragents/explore/.

[21] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with sub word information," Jun 2017. [Online]. Available:https://arxiv.org/abs/1607.04606.

[22] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," Dec 2017. [Online]. Available: https://arxiv.org/abs/1706.03762.