DGA Hunter

Machine Learning – powered weapon for DGA detection

Authors: Filip Bučko, Radek Hranický

NES@FIT research group, Faculty of Information Technology, Brno University of Technology, 2023

Introduction

The rise of Domain Generation Algorithms (DGAs) in cyberattacks poses a serious threat to online security. Detecting and classifying DGAs is crucial to protect against malware, prevent data breaches, and strengthen our cybersecurity defenses. To address the pressing need for Domain Generation Algorithm (DGA) detection and classification, an application that leverages the power of machine learning to efficiently categorize domain names was devloped, enabling users to identify potential threats.

The machine learning model for domain names classification is based on XGBoost (Extreme Gradient Boosting), which outperforms the competition in several key ways:

1. **High Accuracy:** XGBoost consistently delivers superior predictive accuracy, making it a top choice for classification tasks. Its ensemble learning technique combines the strengths of multiple weak models to produce highly accurate results.
2. **Speed and Efficiency:** XGBoost is optimized for speed and efficiency, enabling it to handle large datasets and complex feature sets with ease. This efficiency is crucial for real-time threat detection and classification.
3. **Regularization Techniques:** XGBoost incorporates L1 (Lasso regression) and L2 (Ridge regression) regularization techniques, which help prevent overfitting and improve model generalization, resulting in more robust and reliable predictions.
4. **Flexibility:** XGBoost supports a wide range of objective functions and evaluation metrics, making it versatile for various classification tasks. It can be fine-tuned to meet the specific requirements of domain classification.
5. **Parallel and Distributed Computing:** XGBoost is designed to take full advantage of parallel and distributed computing, allowing it to train and deploy models efficiently on multi-core processors and distributed clusters. This scalability is essential for handling large-scale domain datasets.
6. **Handling the Class Imbalance:** XGBoost also excels in handling the class imbalance problem better than many other machine learning algorithms, because of built-in mechanisms, such as weighted loss functions and subsampling of the majority class, allow it to mitigate the challenges posed by imbalanced datasets effectively.

The machine learning models for DGA detection and classification are built upon a feature set, primarily derived from lexical analysis of domain strings. These features are meticulously chosen for their strong correlation with identifying Domain Generation Algorithm (DGA) domains. The extracted features are listed below:

1. Total domain length - Indicates the number of characters of the domain name.
2. TLD (Top – level domain) Length
3. Length of SLD (Second – level domain)
4. The length of the longest sequence of consonants in SLD.
5. Number of digits in SLD.
6. Number of unique characters in the domain name string (TLD + SLD).
7. Digit ratio
8. Vowel ratio
9. Ratio of consonants
10. Ratio of non-alphanumeric characters
11. Hexadecimal Character Ratio (A-F)
12. Flag of the first digit
13. Publicly known suffix flag
14. Subdomain "www" flag
15. Number of third-level and lower subdomains.
16. Normalized entropy of the chain
17. Dictionary word match
18. N-grams

Processing subdomains in domain names also presents a challenge as relying solely on the second-level domain (SLD) and top-level domain (TLD) for analysis can overlook potential DGA-generated content within subdomains. To address this complexity, we concatenate the entire domain into a single string, enabling more comprehensive feature extraction and enhancing our DGA detection capabilities.

The **classification pipeline (Figure 1)** for domain analysis involves a multi-step process to effectively categorize domains. Here's an overview of how this pipeline works:

1. **Binary DGA Detection Model:**
    - The domain is first passed through a binary classification model designed to determine if it is DGA-generated or not. This model's primary goal is to distinguish between malicious DGA-generated domains and legitimate ones.
    - If the domain is classified as non-DGA (i.e., legitimate), the classification process typically ends here.

2. **Multiclass DGA Family Classification Model:**
    - If the binary model detects that the domain is potentially DGA-generated, it proceeds to the next stage of the pipeline. The domain is then fed into a multiclass classification model, which specializes in identifying the specific DGA family to which the domain belongs. This model can classify domains into one of 93 DGA families.
    - In the user-facing application, the results are typically presented in a user-friendly manner. While the multiclass model can classify domains into 93 DGA families, for simplicity and relevance, only the top 5 DGA families are displayed to user.

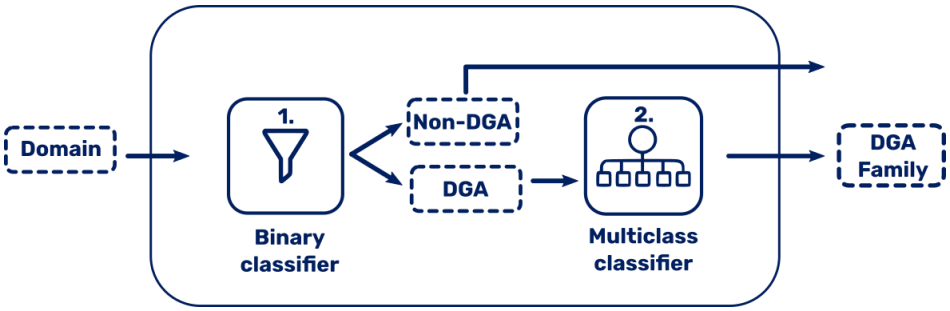

Figure 1: The classification pipeline

In summary, the binary model achieves results with 98% precision, accuracy, and F1 score for DGA detection. In contrast, the multiclass model exhibits slightly lower performance (88%), primarily due to the challenge of class imbalance within the various DGA families it categorizes. Nonetheless, it remains effective in identifying specific DGA families, albeit with a modest reduction in precision.

Getting started – Installation and startup

In this section, this guide will walk you through running a Django application using the requirements.txt file, which lists essential project dependencies. Prerequisites are following:

1. Ensure that you have installed Python on your system
2. Make sure that the port you are going to use is free (in this example the port 8000)

The procedure is as follows:

1. Extract the zip file dga-hunter.zip to your prefered location
2. Open your command-line terminal and navigate to the directory where you've placed the app's source code. You should be in the directory that contains the *requirements.txt* file.
3. Create a Virtual Environment (Optional but Recommended):
    - It's a good practice to create a virtual environment to isolate your project dependencies. In this example we are using conda environemnt (if you don't have installed conda visit https://docs.conda.io/projects/conda/en/latest/user-guide/install/linux.html). To create one, run the following command:

```
conda create --name dga-hunter
```

4. Activate the virtual environment:

```
conda activate dga-hunter
```

5. Install Dependencies. With your virtual environment activated, use the following command to install the required dependencies from the *requirements.txt* file:

```
pip3 install -r requirements.txt
```

6. Start the Django Development Server:

```
python3 manage.py runserver
```

7. Access the App:
    - Open your web browser and navigate to the URL provided by the development server, typically http://127.0.0.1:8000/ by default.
    - You should now see your Django app up and running, accessible through your local development server.

Disclaimer: Django applications should not be run on the development server for production purposes. Instead, use a production-grade web server like Apache, Nginx, or Gunicorn, combined with a WSGI server, to ensure security, performance, and reliability in a production environment.

User Manual

After you successfully managed to run the django backend, simply open the web browser and navigate to http://127.0.0.1:8000/ to access the app (or http://localhost:8000). By default the django app is set to run on port 8000 (This could be changed in settings.py).

In this section, you'll find a comprehensive breakdown of the web app's frontend layout, showcasing the individual elements and explaining their specific functions, enabling users to navigate and interact with the application with ease.

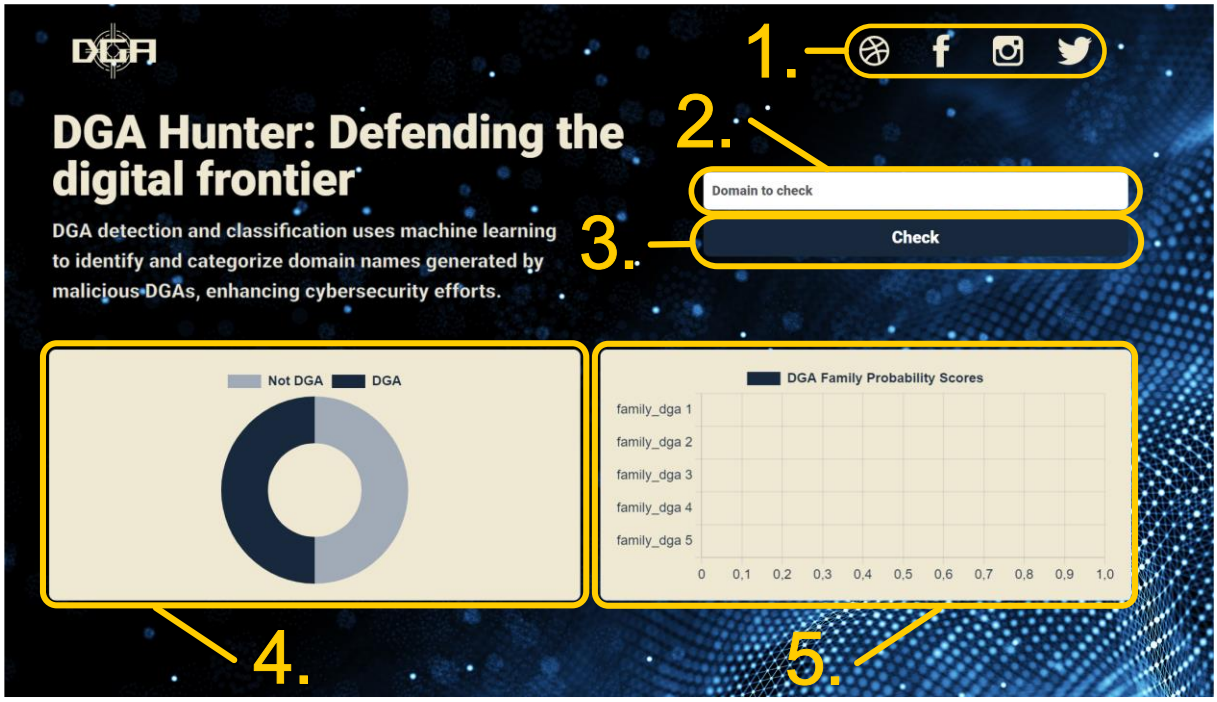

Figure 2: Page layout description

The web application's interface comprises 5 interactive components. The numbers in the following list represent the markings in the **Figure 2**:

1. **Social Links**: links located to our social media profiles.
2. **Domain Input Box:** The input box facilitates domain entry for analysis. Users can input or paste the domain of interest.
3. **Submit Button:** The "Check" button triggers the analysis process and promptly delivers results.
4. **Binary Classification Pie Chart:** Chart illustrates binary classification results. The dark blue color indicates a high probability (90% or more) of a Domain Generation Algorithm (DGA)-generated domain. The grey color indicates domain which was not generated by DGA.
5. **Multiclass Classification Bar Chart:** For domains classified as DGAs, a detailed bar chart displays probability scores for the top 5 DGA families. The absence of bars signifies non-DGA domains, ensuring clear classification information. The DGA families indicate which malware family uses DGA for communication with tha attacker.

The process of analyzing the domain is as follows:

1. Input the domain inside the input box (marked as 2. in the **Figure 2**).
2. Click the “Check” button (3. in the **Figure 2**) to send the domain for analysis.
3. Review the results on the graphs. The example of non-DGA domain classification is presented in the **Figure 3**. The result of the DGA domain classification is in the **Figure 4**

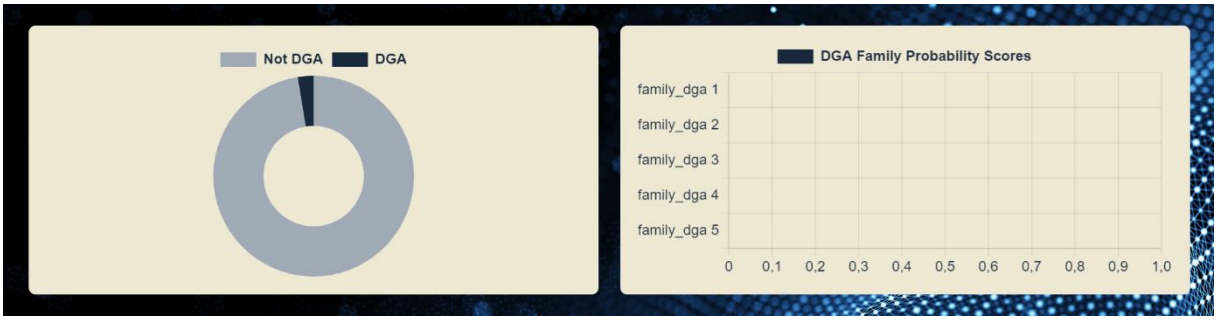

Figure 3: The result of classification of non-DGA domain

Figure 4: The result of classification of DGA domain

Acknowledgements

The development of this software has been supported by the "Smart Information Technology for a Resilient Society" project, no. FIT-S-23-8209 granted by Brno University of Technology.

I want to express my sincere thanks to my fellow research colleagues, namely Ondřej Ryšavý, Kamil Jeřábek and Jan Polišenský, for their assistance, feedback and collaboration during the creation of the AI models.