Web Security - Report

Cesare Corsi, *2156214, Sapienza University of Rome,*
Andrea Massignan, *1796802, Sapienza University of Rome,*

Abstract—In this paper, we introduce **Angel**, a tool developed to enhance web security by intercepting and blocking malicious requests before they can affect the user. Leveraging a machine learning model trained on a dataset of malicious and benign URLs, Angel predicts through a browser extension the threat level of each request and takes preventive actions accordingly. The server can operate within a Docker container to ensure a secure evaluation environment. This proactive approach aims to bridge the gap between threat detection and real-time defense, providing users with an effective tool to safeguard their privacy, security, and resources while browsing the internet.

Index Terms—Malware Prediction; Machine Learning, Network Security; Proactive Security; Privacy, Browser Extensions, JavaScript, Vulnerability; Threat; Security; Confidentiality; Integrity;

1 Introduction

Nowadays a lot of the risks that comes from navigating on the web can harm several aspects of the user, sometimes it is about the user's privacy, or it is about the user's security, or it is about the user's money.
When something malicious is triggered during the navigation, the user can be affected in different ways, for example, the user can have his personal information stolen, his computer can be infected with a virus, it can be used to mine cryptocurrencies or it can be used to attack other computers.

Most of the time the user can not proactively intervene, but they have to deal with the problem after it happens, and sometimes it is too late.
In this paper we introduce **Angel**[1] a tool that intercepts and identifies malicious requests and blocks them before they can harm the user.

This tool relies on a machine learning model that is trained on a dataset of malicious and benign URLs, it is able to predict if a request is malicious or not, and it is built inside a **Docker**[2] container to isolate and provide a better environment to evaluate threats without risking to compromise the host.

2 Background

Before we dive in the description of the tool we need to address some basic knowledge regarding how Extensions and DNN (Deep Neural Networks) works.

An **Extension**[3] is simply put a piece of **Javascript**[4] code that a browser like **Chrome**[5] or **Firefox**[6] can executes in order to achieve some specific behaviour during navigation. The Extensions have access to elements of the **DOM**[7] and thanks to other specific permissions even to the browsing history, rules and so on and so forth.

Manuscript delivered on 06/15/2024

A **DNN** (Deep Neural Network) is a type of Neural Network specialized on a Deep Learning method. Differently from a tipycal NN, the DNNs have more hidden layers between input and output, ensuring more precision and abstraction. In our case we will rely on a CNN (Convolution Neural Network) which is tipycally used in Computer Vision tasks, that will be used to break down strings (URLs) in tokens in order for the model to be able to recognize and correctly predict the outcome.

Last but not least, **Docker** is a particular platform allowing to work in standardized environments using local containers which provides applications and services. The architecture is based on the concepts of **Images** and **Containers**, an image is a template with all the instructions for creating a container, the latter is basically a runnable instance of an image that is almost well isolated from the host OS as it lacks the capabilities to look outside its own process environment.

3 Overview of your proposed approach

In our research we noted that the tools used in identifying threats stopped at detection only, not implementing a real security system. We tried to combine various aspects of those studies to fill the gap between detection and actualization.

First of all we needed to choose a model that best fits our purpose, we considered using a Logistic Regression

1. https://github.com/duchannes19/Angel
2. https://www.docker.com/
3. https://developer.chrome.com/docs/extensions
4. https://developer.mozilla.org/en-US/docs/Web/JavaScript
5. https://www.google.com/intl/it_it/chrome/
6. https://www.mozilla.org/it/
7. https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model

model to test the validity of the **dataset**[8] that was used. The original dataset is a collection of approximately 651.191 URLs labeled as: benign, malware, phishing and defacement. The last two labels were merged in the malware category in order to have a binary outcome to simplify the procedure.

After that we chose a CNN that was trained on this type of dataset that follow this kind of architecture:

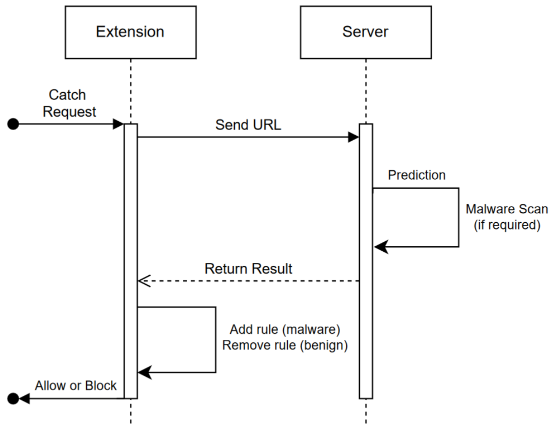
- Embedding Layer
- Convolutional Layer
- Max Pooling Layer
- Fully Connected Layer 1
- Fully Connected Layer 2
- Dropout Layer
- Activation Function: ReLu

In order to pass the URLs to the CNN we preprocessed the dataset by firstly converting the categorical labels into numerical format using a label encoder, we separated the features (URLs) and labels (types), we used a pretrained **Tokenizer**[9] to tokenize the URLs by iterating over them, removing the prefixes (http/https and www) to standardize the format, then we converted them into token Ids and lastly combine and transform them into a Pytorch tensor.
The model is then trained and the result saved as a **pth** file to be later used inside our application.

Now that we have our trained model we can use it inside a simple **Python**[10] application, in our case a **Flask**[11] server, that handles incoming requests from the extension, evaluates the risk with a prediction from the CNN (and optionally download and scan a file if present in the URL) and return the result back to the latter. But the server also offers other important functionalities, for example we introduced a whitelist system for predictions that the users consider safe, and this also gives us the possibility for the model to be trained again with the user custom preferences.

The frontend of our application, as we said in the introduction, is a browser extension, specifically a Chrome one. The extension has the ability to intercept the request sent on the searching bar, during this time it is passed to the server for analysis and if the result is malicious then: a black screen with a security warning is overlayed on top of the page, rules to block every request from that page are inserted in the Chrome rule policy and a button is displayed to grant the user the possibility to continue the navigation nonetheless by adding the URL to the whitelist.

8. https://www.kaggle.com/datasets/sid321axn/malicious-urls-dataset/data
9. https://huggingface.co/docs/transformers/main_classes/tokenizer
10. https://www.python.org/
11. https://flask.palletsprojects.com/en/3.0.x/



What happens specifically when a user makes an URL request is:

- The request is captured by the extension **contentscript** and sent with a message to our **service worker**, the latter then sending an HTTP request to our local Flask server.
- The server receives the request, it breaks down the URL by removing the prefixes to adapt it to the needs of the model, then it is tokenized in the same way that happened during the training phase.
- If the URL it is found inside the **Whitelist** database, then the procedure is interrupted and immediately returned as 'benign' back to the extension.
- Otherwise A prediction is made, if the URL is considered a malware it is also checked if it contains a file based on the last part of the string.
- If it is a file it is then downloaded and analyzed with a simple **malware scanner**[12] implemented to work within our environment.
- The results are then embedded inside a json response and sent back to the **service worker** that handles the consequential actions depending on the outcome.
- If the prediction is 'benign' then no severe actions are performed, if the URL was already flagged and blocked inside the **Rules** then it is removed and allowed.
- If the prediction is 'malware' then the page is flagged by adding a **Rule** with the **chrome.declarativeNetRequest API**[13] in order to block network requests.
- Additionally an overlay is shown with a button with the option to add the URL to the **Whitelist**.

4 Evaluation

During the training and evaluation phase of our CNN model we obtained the following results:

- Epoch [1/10] Loss: 0.2551, Accuracy: 0.8881
- ...

12. https://github.com/password123456/malwarescanner/tree/main
13. https://developer.chrome.com/docs/extensions/reference/api/declarativeNetRequestin

- Epoch [10/10] Loss: 0.0927, Accuracy: 0.9622

These results are promising, even though we are using a small dataset for our machines capabilities. To achieve a more consistent result we could use a larger dataset but that would require a lot more VRAM in order to do so.
In a more idealized scenario we imagined that if this model could be trained on a larger quantity of URLs features the result could lead to an optimal version for the use that we intended and to evaluate the real performance of such model we would need to test it on a lot of URLs that were not inside our dataset.

When we open the Chrome browser and load in it our extension, we tested the response time between the catching of the request and the elaboration on the server that runs our model. This happens in a matter of one or two seconds, in some cases even less, this depends on the level of delay that occurs when our extension needs to inject the overlay content in the Chrome tab.
To precisely evaluate this kind of delay we could implement a timer inside the extension, but this could prove challenging given the way the extension communicates with the browser through the **Chrome Message Passing API**.

Nonetheless the results were better than we have hoped.

5 Related work

Contributions for our work comes mainly from the first paper [1] we studied. Here we learned that by using a Machine Learning approach, such as DNN, we could achive a higher performance and reliability for what it concerns the detection of malicious requests. But in their specific case they refrained from implementing an actual defense response.

For this motive we thought of combining the other approach that is operated in the second paper [2] even though their objective was different than ours. Here the concept of using an extension in order to have more control on what happens inside the browser gave us the perfect opportunity to test it with our own purpose.

6 Conclusions

Our research revolved around the development of **Angel**, a browser extension designed to intercept, identify, and block malicious requests which can possibly damage and corrupt the users' system.

By itself, the application works adequately, although some improvements can be made. For example, instead of relying on a whitelist, at some point the model could be entirely retrained with the sum of URLs that the user knows are absolutely safe. Since this requires a lot of power and resources, perhaps a cloud service could perform this task and forward the updated model to the user.

Another important aspect to consider is that the model should be first trained on an enriched dataset that could comprehend an incredibly large and reliable amount of data, like the type of the request and all the other essential parameters inside its body, in order to perform better in its initial steps.

References

[1] M. Sharif, J. Urakawa, N. Christin, A. Kubota, and A. Yamada, "Predicting impending exposure to malicious content from user behavior," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, (New York, NY, USA), p. 1487–1501, Association for Computing Machinery, 2018.

[2] Q. Chen and A. Kapravelos, "Mystique: Uncovering information leakage from browser extensions," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, (New York, NY, USA), p. 1687–1700, Association for Computing Machinery, 2018.