AN ARTIFICIAL INTELLIGENCE URL PARSER FOR SAFER WEB BROWSING AND DETECTION OF SUSPICIOUS LINKS

James Jin[1] , Gayatri S[2] and Yu Sun[3]

[1]Crean Lutheran High School, Irvine, CA 92618
[2]University of California, Irvine, Irvine CA 92697
[3]California State Polytechnic University, Pomona, CA, 91768

ABSTRACT

With more than seven billion people actively using the Internet, the number of cyber attacks has increased, and personal data breaches have become a concern among the general public. The COVID-19 pandemic has only increased the use of online platforms and services for work and leisure activities, which opens the door to more scams, viruses, and other cyber security breaches. Guided by SEO techniques and research regarding dangerous website and domain patterns, we have designed and implemented a visual system that tracks suspicious links on an active webpage and marks them in order to alert users to proceed with caution. Our AI utilizes linear regression to best detect trends in URL parsing, comparing them with registered unsafe links to see if they pose similar threats. The results reveal that AI isn't entirely accurate since some trends are hard to decipher; however, it can reliably flag certain redirects and out-of-domain links that would otherwise remain hidden to users.

KEYWORDS

Safe web browsing, Security application development, Phishing prevention

1. INTRODUCTION

The topic of cyber safety has been at the center of attention in the recent past, yet disturbingly, there has been little to no light shone on many of the millions of malicious links currently on the internet [1, 2, 3]. Information about links is often obtained through rumors and reports, and therefore thousands fall victim to malicious links before they are detected and removed. I noticed this pattern in 2020 and early 2021 when my research team and I received several suspicious links as we developed this project. We managed to avoid these links due to kind warnings posted online by other victims of such links. Therefore, SafeLink strives to help potential victims identify and eliminate dangerous links before they can be harmed by them. SafeLink serves as a wall or a buffer between malicious links and innocent users. Currently, viruses are evolving just as fast as anti-virus software, which makes phishing links especially dangerous [4, 5]. The goal of SafeLink is to identify and neutralize problems before they occur, while anti-virus software is used to destroy threats after users have fallen prey to an attack. SafeLink does not forcibly remove links from a users' searches, but instead warns users that what they are about to do may have consequences. SafeLink is not supposed to be used as the ultimate defense against all viruses, but rather as a first line of defense that can hopefully reduce the possibility of needing to use anti-virus software.

With certain links posing a problem, search engines have developed filters to get rid of such links [6, 7]. Functions such as safe-search and advanced search usually only show trustworthy sources

[8, 9]. The problem with such searches is that they often have limited content and are sometimes too specialized for just browsing the net. These searches assume that users are looking for a specific topic, so they are not optimized for broader answers. Advanced Search by Google is usually used for academic purposes, while many websites that are actually useful are blocked by Bing's Safe Search function [10]. The functions of such algorithms are flawed in being too careful, often yielding search results that either have nothing to do with the topic searched, or are older and/or summarized. Such searches have many functions, but many either pay no attention to them or do not know how to use them. Another alternative is to use Chrome plugins made by anti-virus companies [11]. The problem there is that they are normally used to detect harmful files only after users have downloaded them, allowing the companies to then request users to download their anti-virus software. These extensions are often terribly outdated, confusing to use, or outright useless. Many of the most effective extensions used for malicious link detection are locked behind paywalls. Unlike Chrome extensions designed to advertise other anti-virus products, SafeLink was designed for everyone with no paywall. That is where SafeLink differs from the majority of existing options.

The solution we came up with is SafeLink. Our goal was to identify and eliminate potentially threatening links before they can harm users. Our approach was inspired by early antivirus software, which merely identified suspicious files instead of letting users decide whether to delete or leave them. One beneficial feature of SafeLink is that it allows users to choose which links to click. SafeLink doesn't remove potentially suspicious links, but instead lets users choose, since links may be incorrectly identified as harmful when in fact they only have excessive redirects or advertisements. The SafeLink team is also working toward a blacklist system where if links are identified as dangerous, users can add them to this list and they will not show up again. SafeLink is not meant to be a complete solution for all online problems, but rather a first line of defense to complement the features already offered by antivirus software. Its target users are children and older adults unfamiliar with technology, since the younger generation may already know how to avoid suspicious links [12].

In two application scenarios, we demonstrated how the techniques given above can benefit the users of this app. First, we demonstrated the usefulness of our approach via a case study on the evolution of AI technology using the spam resistor application. Second, we analyzed the evolution of the spam detector and its methods of detecting and blocking spam calls. We conducted extensive research on past applications that had code similar to our application, and explored the possibilities of using this code. We eventually came to the conclusion that the AI should learn from scratch in order to better accommodate users' demands. It should also use a simple system that allows for an easy understanding of how it works. This allows the AI to grow smarter as the user uses the program more and more. This also makes it so that the AI is not as efficient when it starts, however. It also made us realize that most spam detectors use methods such as checking for certain kinds of links and/or the length of the links. This helped us evolve our own rating system toward targeting longer or shorter links, or links with many redirects. Even with this information, the code did not run well, which led us to download suitable libraries fitted to trim and reorganize code so that it can mimic the effectiveness of professional extensions.

The remainder of this paper is organized as follows: Section 2 gives details and information on the challenges we met during the experimental and design phases; Section 3 focuses on the details of our solutions, which correspond to the challenges given in the previous section; Section 4 presents the relevant details of the various experiments we did, followed by related research in Section 5. Finally, Section 6 provides concluding remarks, as well as the mapping and planning of future work on this project.

2. CHALLENGES

In order to design and implement a visual system that tracks suspicious links on an active webpage and marks them in order to alert users to proceed with caution, a few challenges were identified as follows.

2.1. Challenge 1: Utilizing more than one type of Code

The main challenge of this project was to utilize more than one type of code. We had to use json, html, Java, and Python for the extension to work. Out of these scripts, we were familiar with json and Python, but had less knowledge of html and Java. This caused our overall progress to slow down as we had to learn both scripts as we advanced through the coding process. This caused many unseen problems as we had to adjust our plans, requiring hours of fixing and readjusting code. For example, when we learned Java doesn't run in order, this created over two hours of extra work culling through code libraries to find a few lines to solve the problem out of hundreds of scripts.

2.2. Challenge 2: Extent of the Initial Research

Another challenge we faced was the extent of the research we needed to do on the topic at hand. As a team, we had comparatively little experience coding large projects, chrome extensions, or security software in general. This forced us to dedicate significant time to researching potential solutions on similar projects and searching for viable extension code options. This helped us build a decently effective AI, but with room for improvement and growth. The problem with AI machine learning is that it isn't effective in the beginning, hence the name machine learning, not machine expert. This aspect was somewhat troublesome, since we had to teach it what was correct and what wasn't. Only after significant time and research did we produce a more effective AI that would make fewer mistakes.

2.3. Challenge 3: No Prior Formula to Define Suspicious Links

One other challenge was the fact that there is not an accurate formula already in use to define suspicious links. There are patterns used to recognize suspicious links, but no one central feature that can easily differentiate suspicious and non-suspicious links. Some features that are associated with suspicious links are aspects such as redirects or less trustworthy endings. However, such features do not definitively define suspiciousness, since plenty of non-suspicious links also use the same endings or utilize redirects. One example is Gmail, which relies on redirects for site navigation. Another example is the use of ".net" by various game and development companies that use it either because it costs less or the .com version was not available. Even these associative features do not allow the AI to be error free, as there are more non-suspicious websites than suspicious ones.

3. SOLUTION

SafeLink is a phishing link detector that uses several factors and known signs of phishing links, along with a rating system to detect whether a link is safe or unsafe. There are some common attributes that are used to identify suspicious links. These include shortened links, out-of-domain links, links with redirects, links that end in less credible endings, and links that contain certain words. This helps SafeLink identify general links, while remaining unbiased towards normal links (with some exceptions). SafeLink uses basic spam detection techniques along with a unique and customized rating system originally designed for use in Google Chrome extensions. Having it structured in a Chrome extension allows it to have access to active tab information and allows the

backend to integrate with the web server more easily. SafeLink uses the spam detection system to scan all current active links on the user's webpage to identify any suspicious ones and warn the user against them. It also utilizes AI machine learning, so even if the system is imperfect at the start, after enough use and time it can achieve more accuracy. The rating system can also easily be expanded to include factors such as the length and character counts of the links.

Hence, SafeLink has three main layers:

- The Google Chrome extension, which reads and parses the links on the page and sends them to the backend using POST or GET requests.
- The background layer, which helps transfer data from the webpage to the Flask server in the backend.
- The Flask Server, which houses the AI algorithm and sends back a prediction for the safety of the link based on its training and rating system. The flow of these layers per design is illustrated below. It is a bidirectional workflow, which means it goes from frontend to backend and vice versa.

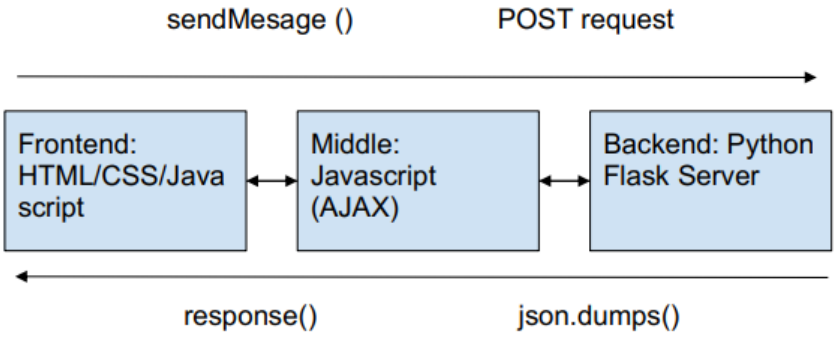

Figure 1. The three layers, with the function calls linking one direction on top and the other on the bottom. This end-to-end connection scrapes the links from the page and processes them on the Python server, sending back information to highlight tags.

SafeLink's main components include the link parser and AI model, which is implemented in a Chrome extension where it can easily access and scan webpages. It uses web-scraping techniques in order to receive information on the page and focuses only on certain aspects such as <a> tags. It is a Google chrome service only, and therefore limited to only Google. Its AI model is quite simple and effective when improved and put to use. It identifies the links from a webpage and adds them to an organized dictionary. Afterwards, it will rate the links based on certain attributes to determine the credibility of the links. The attributes are inside functions, which quantify them for use. Each link receives a set of values based on the quantified attributes and predictions are then made with the support of the vector machine algorithm provided by Sklearn, which adds data to the dataset for the machine to learn from. To make the code function, we used many tools and services such as Python ML and web hosting libraries (e.g., Sklearn, Gunicorn, etc.) as well as a python flask server and SVM AI model object. The extension also features end-to-end connection, with Javascript AJAX sending data based on certain triggers between the pages and the data that is collected and stored in dictionaries on both ends to be flagged as needed.

1. Link Parser

    a. Implementation

        i. Chrome extension format

ii. Javascript web-scraping

1. Focuses on collecting <a> tags and sending them to the background script.

2. Once "unsafe" links have been identified by the other components, it matches them up to the tag using the dictionary saved during the first <a> parse and uses the highlight function to show it on the page

iii. Images:

Tag highlighting function (Figure 2):

```
function highlight(tag) {
    if (tag){
        var h = tag.innerHTML;
        var s2 = '<span style="background-color: aqua;">' + h + '</span>';
        tag.style.backgroundColor = "aqua";
    }
};
```

Figure 2.

Collecting pages <a> tags to store in the dictionary using the name of the links as the keys, and tags the values (Figure 3):

```
var dom = document.domain;
var aTag = document.getElementsByTagName("a");

var badLinks = [];
badLinks.push(dom);

var linkDict = {};

for (let i = 0; i < aTag.length; i++){
    var link = aTag[i].href;
    console.log(link)
    linkDict[link] = aTag[i];
    badLinks.push(aTag[i].href);
```

Figure 3.

Message handler sends request (Figure 4):

```
chrome.runtime.sendMessage({message: msgString}, async function(response) {
```

Figure 4.

b. Tools/Services

i. Google chrome extension (tool/service)

ii. AJAX POST requests

2. AI Model

a. Implementation

i. Gets links from frontend and organizes them into a dictionary.

ii. Identifies links based on attributes that determine credibility.

iii. Makes functions to quantify attributes—for example, if checking for an out-of-domain link, it will return a 5 for one end, and 1 for the other. For non- binary items, it would be 1-5, while binary would be 1 and 5 to maintain a scalable system (Figure 5).

```
def ood(link: str, dom: str):
    name = dom.split('.')[1]
    if name in link:
        return 5
    return 1
```

Figure 5.

iv. Gives each link a set of values using the functions mentioned above.

v. Makes a prediction using the support vector machine algorithm provided by Sklearn and adds this data to the dataset in order to make the model learn (Figure 6).

```
rec_model = svm.SVC()
rec_model.fit(data, s)

badLinks = []
for link in links:
    values = [bitly(link), redirect(link), blog(link), ood(link, domain), endDomain(link)]
    print(values)
    link_safety = rec_model.predict([values])
    data.append(values)
    s.append(link_safety[0])
    print(link,link_safety[0])
    if link_safety[0] == 'unsafe':
        badLinks.append(link)
```

Figure 6.

b. Tools/Services

i. Python ML and web hosting libraries (Sklearn, Gunicorn, etc.)

ii. Python Flask app server

iii. SVM AI model object

3. End-to-end connection

   a. Javascript AJAX sends data based on certain triggers between the pages and this data is collected and stored in dictionaries on both ends to be flagged as needed (Figure 7).

```
chrome.runtime.onMessage.addListener(
    function(request, sender, response){
        $.ajax({
            url: "https://safe-link-heroku.herokuapp.com/",
            type: "POST",
            data: request,
            success: function(resp){
                console.log(resp);
                response(resp)
            },
            error: function(er,a,b){
                console.log("error has occurred");
            }
        });
        return true;
    }
);
```

Figure 7.

4. EXPERIMENT

The experiment that was run included three people, one with an older computer, one with an Apple computer, and one with a fully updated windows system. The results among the three were similar as Chrome doesn't differ with different operating systems. While this is a small number of participants, since the nature of this is slightly more objective, it would still yield results that are useful to improve and test the extension. There were enough participants to determine some basic issues and provide basic feedback. Each participant was given a list of websites to rate as "safe" or "unsafe" and the results were matched with the majority's consensus for that website's credibility. The feedback consisted mostly of the rating system and AI, to the extent that it could function, as the experiment did not carry on long enough for the AI to make improvements. SafeLink worked on more obvious websites, but wasn't as helpful on others, according to the feedback from participants. The feedback/results suggested that there needs to be an overhaul of the AI to make it operate more precisely, while covering more topics.

There were several websites that were able to be correctly identified as "safe," but since it's hard to define "unsafe," a lot of "unsafe" websites weren't correctly identified as such, since no absolute rules could be applied to them (see Table 1).

Table 1. Several links and their predicted versus actual safety

Links	Predicted	Actual
www.coolmathgames.com	safe	safe
www.discord.com	safe	safe
www.youtube.com	safe	safe
www.quizlife.com (on CoolMath)	unsafe	unsafe
www.bit.ly/privateroom	unsafe	unsafe
www.smbgames.be/	unsafe	safe
http://fulldownload2pcgames.blogspot.com/	unsafe	safe
www.bitly.com/WICS	safe	unsafe

Based on feedback, the system was changed to include a rating for the length of the link and how many characters were present. The AI was also further trained so that it would not make as many mistakes in the beginning. This meant that certain generalized rules were narrowed by adding conditional statements. The experiment was then redone with the same websites and conducted in a similar way, where participants decided if they thought a website was “safe” or “unsafe” and this information compared to the AI results from the experiment.

Once again, definitions were decided by the participants. Most participants predicted the same credibility score(s) for the websites, but that was still something that was meaningful to pick apart because it helped further define what is “safe” and what is “unsafe” (see Table 2).

Table 2. More links and their predicted versus actual safety

Links	Predicted	Actual
www.coolmathgames.com	safe	safe
www.discord.com	safe	safe
www.youtube.com	safe	safe
www.quizlife.com	unsafe	unsafe
(on CoolMath)		
www.bit.ly/privateroom	unsafe	unsafe
www.smbgames.be/	unsafe	safe
http://fulldownload2pcgames.blogspot.com/	unsafe	unsafe
www.bitly.com/WICS	safe	unsafe

During the experiment, we were able to get good feedback on what aspects of the AI worked and which ones needed more conditions to be effective. Out-of-domain links and links with inappropriate wordings were flagged, but other attributes were either incorrectly flagged or not flagged at all when they should have been, so we tried to narrow the categorization further by using both the feedback and results. This process of improvement doesn’t entirely solve the challenges, and alludes to the fact that link safety recognition/deciphering doesn’t have a formulaic answer and is still being researched. Since a lot of cases have possible good and bad examples, knowing what situations created what results was something that came up a lot during the experiment process. We were able to correctly identify out-of-domain links and blatantly dangerous ones, but many of these possessed the same characteristics as “safe” websites. From this fact, we realized we would need more data and several new rules/conditions to allow the AI to respond the way it should.

5. RELATED WORK

Alexandros Ntoulas, et al. presented a system to detect potential spam websites through content-based analysis. By analyzing content within links, including length of the page and the amount of anchor text within the links, the system predicts the possibility of the link being spam. While we used Flask Server to house the AI algorithm and train the system to learn, Alexandros utilized the C4.5 classification algorithm [13].

Chun-Ying Huang, et al. discussed an approach that detects phishing websites by generating text-image combined signatures and matches them with those stored in the database. This approach is more effective for websites of large institutions, such as banks and shopping websites [14].

Hyunsang Choi, et al. presented an approach to not only detect potential malicious links, but also identify the type of attacks users could get from those links. Knowing the attack types, users will know how to react if they accidentally click on such links, or choose to visit the websites anyway if the websites are spam [15].

6. CONCLUSION AND FUTURE WORK

In summary, the goal of SafeLink is to help users quickly detect potential malicious links when they browse, and to provide them with warnings before they click on such websites. By summarizing common features of malicious links and coding these features as attributes into its system, SafeLink is able to determine the safety level of links and provide users with warnings about malicious ones. At the same time, developers can easily add new attributes to its grading system over time. With AI

machine learning, the system is able to learn from itself and improve its accuracy over time. Experiments on the effectiveness of AI machine learning shows that while it is not effective within a short period of use, it can be just as effective, if not more so, than normal non-learning AI systems over time. With the core difference between AI Learning and normal AI being the level of aptitude when the AI is first made, AI machine learning doesn't need a developer to completely re-code or shuffle through hundreds of lines of scripting to improve, while a normal AI does need regular updating to maintain effectiveness and make improvements. This makes it so that AI machine learning stands out in terms of both saving time and its effectiveness over long-term use. Of course, a pre-planned AI would be smarter in the beginning, something akin to comparing a robot with a child. However, as the AI machine learns, it will surpass the pre-planned AI, which is proven by both this experiment and the fact that many companies are switching to more advanced versions of AI machine learning and replacing pre-planned AI systems that have reached their limits for learning.

Although practical, the current AI of SafeLink could still be improved upon. The accuracy of the AI in the beginning is relatively low, but this will only increase over time. This is a major issue, however, as it will not be of as much help in the beginning, and may also deter users from unharmful websites. This learning curve also raises the possibility of an unsafe website tricking the AI into thinking it is legitimate. The AI, although crude, can be optimized via a slow process of teaching it what is correct and what is incorrect. Although after optimization the AI will definitely be better, it still will take a long period of use before a higher accuracy can be achieved.

The current AI system has a large potential for growth. Although it may not be able to reach complete accuracy, with enough training in code, it can be greatly improved. It can also be

improved by adding new factors to determine scam links to make sure that no suspicious links sneak past its detection.

REFERENCES

[1] Alhayani, Bilal, et al. "Effectiveness of artificial intelligence techniques against cyber security risks apply of IT industry." Materials Today: Proceedings (2021).

[2] Gupta, Brij B., ed. Computer and cyber security: principles, algorithm, applications, and perspectives. CRC Press, 2018.

[3] Berman, Daniel S., et al. "A survey of deep learning methods for cyber security." Information 10.4 (2019): 122.

[4] Solé, Ricard, and Santiago F. Elena. Viruses as complex adaptive systems. Vol. 15. Princeton University Press, 2018.

[5] Iliev, Anton, et al. Some models in the theory of computer viruses propagation. LAP LAMBERT Academic Publishing, 2019.

[6] Sonowal, Gunikhan, and K. S. Kuppusamy. "PhiDMA–A phishing detection model with multi-filter approach." Journal of King Saud University-Computer and Information Sciences32.1 (2020): 99-112.

[7] Jain, Ankit Kumar, and Brij B. Gupta. "Towards detection of phishing websites on client-side using machine learning based approach." Telecommunication Systems 68.4 (2018): 687-700.

[8] Rahim, Ifra, Humaira Mushtaq, and Shabir Ahmad. "Evaluation of Search Engines Using Advanced Search: Comparative Analysis of Yahoo and Bing." Library Philosophy and Practice (2019): 1-12.

[9] Halevi, Gali, Henk Moed, and Judit Bar-Ilan. "Suitability of Google Scholar as a source of scientific information and as a source of data for scientific evaluation—Review of the literature." Journal of informetrics 11.3 (2017): 823-834.

[10] Martín-Martín, Alberto, et al. "Can we use Google Scholar to identify highly-cited documents?." Journal of informetrics 11.1 (2017): 152-163.

[11] Chandru, N. "A Review on Phishing Attacks and Anti-Phishing Browser Plugins."

[12] Jiang, Mengtian, et al. "Generational differences in online safety perceptions, knowledge, and practices." Educational Gerontology 42.9 (2016): 621-634.

[13] Ntoulas, Alexandros, et al. "Detecting spam web pages through content analysis." Proceedings of the 15th international conference on World Wide Web. 2006.

[14] Huang, Chun-Ying, et al. "Mitigate web phishing using site signatures." TENCON 2010-2010 IEEE Region 10 Conference. IEEE, 2010.

[15] Choi, Hyunsang, Bin B. Zhu, and Heejo Lee. "Detecting Malicious Web Links and Identifying Their Attack Types." WebApps 11.11 (2011): 218.