IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 10, 2020, accepted April 24, 2020, date of publication April 30, 2020, date of current version May 15, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2991403

PhishHaven–An Efficient Real-Time AI Phishing URLs Detection System

MARIA SAMEEN[1], KYUNGHYUN HAN[2], AND SEONG OUN HWANG[3], (Senior Member, IEEE)

[1]Department of IT Convergence Engineering, Gachon University, Seongnam 13120, South Korea
[2]Department of Electrical and Computer Engineering, Hongik University, Sejong 30016, South Korea
[3]Department of Computer Engineering, Gachon University, Seongnam 13120, South Korea

Corresponding author: Seong Oun Hwang (sohwang@gachon.ac.kr)

This work was supported by the National Research Foundation of Korea (NRF) Grant funded by the Korea Government (MSIT) under Grant 2020R1A2B5B01002145.

ABSTRACT Different machine learning and deep learning-based approaches have been proposed for designing defensive mechanisms against various phishing attacks. Recently, researchers showed that phishing attacks can be performed by employing a deep neural network-based phishing URL generating system called DeepPhish. To prevent this kind of attack, we design an ensemble machine learning-based detection system called PhishHaven to identify AI-generated as well as human-crafted phishing URLs. To the best of our knowledge, this is the first study to consider detecting phishing attacks by both AI and human attackers. PhishHaven employs lexical analysis for feature extraction. To further enhance lexical analysis, we introduce URL HTML Encoding to classify URL on-the-fly and proactively compare with some of the existing methods. We also introduce a URL Hit approach to deal with tiny URLs, which is an open problem yet to be solved. Moreover, the final classification of URLs is made on an unbiased voting mechanism in PhishHaven, which aims to avoid misclassification when the number of votes is equal. To speed up the ensemble-based machine learning models, PhishHaven employs a multi-threading approach to execute the classification in parallel, leading to real-time detection. Theoretical analysis of our solution shows that (1) it can always detect tiny URLs, and (2) it can detect future AI-generated Phishing URLs based on our selected lexical features with 100% accuracy. Through experiments, we analyze our solution with a benchmark dataset of 100,000 phishing and normal URLs. The results show that PhishHaven can achieve 98.00% accuracy, outperforming the existing lexical-based human-crafted phishing URLs detection systems.

INDEX TERMS AI-generated phishing URLs, ensemble machine learning, human-crafted phishing URLs, lexical features, multi-threading, tiny URLs, URL HTML encoding, voting.

I. INTRODUCTION

The distinctive characteristics of machine learning, ranging from detecting and extrapolating patterns to adapting a new environment, enable it to be a crucial part of technological systems like nuclear power plants monitoring, cyber and homeland security, computer vision, and IoT(Internet of Things), to name a few. In [2], the authors demonstrated through their study that machine learning is effective in providing security for IoT based systems. With the increasing demand of security, machine learning-based systems usually outperform traditional humans-based security monitoring systems. Today, when the world heavily relies on electronic communications, connected devices lead to a variety of online threats and cyber attacks every day. In [3], the authors discussed in detail how cyber attacks for smart grids can be carried out in different phases and forms. In [4], the authors highlighted how cyber attacks on load forecasting can affect the crucial operational decisions needed for electricity delivery. In [5], the authors focused on FDI(false data injection) attacks and how to mitigate such cyber attacks. And in [6], the authors investigated the effects of cyber attacks on power grids.

Among a wide range of online threats and cyber attacks, phishing is the most common one. Phishing attack is any fraudulent attempt that involves an activity of disguising

The associate editor coordinating the review of this manuscript and approving it for publication was Fuhui Zhou.

oneself as a trustworthy party to obtain sensitive information. Phishing attacks can be of different types which include E-mail spoofing, website forging, social engineering, etc. One of the subtle yet deceiving methods to perform phishing attacks is phishing URLs. Phishing URLs are types of URLs which are especially crafted by phishing attackers. The common characteristic of these URLs is that they appear to be a legitimate URL but redirect users to the attackers' websites. According to the report published by APWG(Anti-Phishing Working Group) on November 4, 2019 [7], the number of phishing attacks has risen to a high-level which were not seen since late 2016. Their report discussed and demonstrated the highest level of phishing attacks carried out throughout the year of 2019 based on 3 quarters.

Several researches have been conducted to prevent, mitigate and even to correct phishing attacks. Majority of the researches are focused on using different machine learning models, deep learning models and/or the combinations of the models. In [8], the authors performed a study in detail on how to build potential cybersecurity systems using machine learning. Researchers and security analysts tend to improve phishing URLs detection systems through machine learning and deep learning models. In [9], the authors studied the optimization of phishing URLs detection systems through genetic algorithms. Also in [10], the authors designed a phishing URLs detection system. While over time, phishing adversaries have also spanned their horizons(i.e. targeting different end-devices) and enhanced their attacking strategies. In [11], the authors conducted a study to highlight the phishing attacks performed on mobile devices along with defence mechanisms and existing challenges. In [1], the authors proposed a model named as **"DeepPhish"** which is specially designed to generate AI phishing URLs. DeepPhish [1] takes Simple Phishing URLs, i.e., human-crafted phishing URLs as its input and generates new phishing URLs. Majority of these newly generated phishing URLs, i.e., AI-generated Phishing URLs are capable enough to easily bypass existing prevalent phishing detection systems. With this, the near future of cyber attacks can be easily forecasted where AI will be used to carry out highly sophisticated malicious attacks, known as **"Offensive AI"**. A report by DARKTRACE [12] showed how a new paradigm of cybersecurity threats will emerge with AI driven attacks, enabling attackers to incorporate the characteristics of AI such as impersonating the trusted users, mimicking the users' behaviors, autonomous decision making ability, etc. along with existing sophisticated attacks and malwares. In [13], the authors critically examined ''machine ethics'' and concluded that machine ethics is not an appropriate technological fix to the social problems arising due to AI applications.

Furthermore, machine learning and deep learning models are primarily crafted statistical models to perform specific tasks effectively without any external instructions, they still lack accuracy in performing those specific tasks, resulting in misclassifications. There can be multiple reasons for lacking accuracy in a performance, e.g., mis-labeled data, inappropriate features reduction or selection, over-fitting or under-fitting of features. One of the most important reasons behind this lack of performance is the models' architecture restrictions. That is, the internal structures of models restrict models to manipulate and analyze different types of features. For example, Linear Regression models perform very well for features or patterns having linear relationships among them, but perform poorly when there are non-linear relationships. In [14], the author demonstrated various limitations of three different types of Boltzmann machine learning procedures. Due to this, even if we may facilitate models with a) ample amount of datasets, b) perform proper feature reduction or selection process, c) avoid over-fitting or under-fitting, models still somehow fall short of generating accurate results as models are unable to cater different types of features.

To address the above-mentioned problems, we propose **PhishHaven**, an efficient real-time AI-generated Phishing URLs detection system. Our study of relevant literature shows that PhishHaven is the first phishing detection system designed to detect AI-generated Phishing URLs. PhishHaven is especially designed to detect phishing URLs generated by DeepPhish [1]. Our proposed system uses lexical features-based extraction and analysis techniques. To proactively detect and classify a URL on-the-fly, we additionally introduce URL HTML Encoding as a lexical feature to further boost PhishHaven. In addition to this, we introduce URL Hit, an approach to effectively detect tiny URLs. Furthermore, we also design a new paradigm for executing ensemble-based machine learning for PhishHaven. This new paradigm makes parallel execution of ensemble machine learning models using multi-threading approach for training and testing phases. PhishHaven also employs unbiased voting concept in decision-making process to assign final labels (i.e., either phishing or normal) to the URL(s).

Main contributions of this study can be summarized as:

1) We propose the first AI-generated Phishing URLs detection system which is capable of detecting phishing URLs generated by DeepPhish [1] with high precision, accuracy and F1-measure of 98%. Even for Simple Phishing URLs, it outperforms the other existing detection systems with higher precision.
2) We introduce URL HTML Encoding as an additional lexical feature to classify URLs proactively and on-the-fly.
3) We introduce a URL Hit approach which can detect any tiny URL that can be from either phishing or normal category. Our approach is completely independent of any URL shortening softwares, algorithms, methodologies and does not require any prior knowledge in this regard.
4) We propose a new paradigm of execution for ensemble machine learning, which is comprised of parallel execution of ensemble-based machine learning models through multi-threading. Parallel execution in training and testing phases speeds up processes, hence allows to detect phishing URLs in real time.

5) The proposed detection system boasts various desirable features. First, it is independent of any third-party services (i.e., WHOIS, Team Cymru, etc.) because all the procedures including features extraction from a URL, examination and classification of a URL are performed within our detection system. Second, it is independent of languages because it analyzes URLs only. And third, it is capable to detect zero-day attacks because our detection system analyzes URL based on URL's lexical features.

The rest of the paper is organized as follows: Section II discusses our main motivation. Section III provides a brief overview of different approaches used for designing different phishing URLs. Section IV presents our proposed solution along with its methodology and time complexity analysis in detail. Section V highlights the experiments and evaluations of our proposed solution. Section VI describes some related work. Finally, Section VII concludes this paper along with future work and directions.

II. MOTIVATION

We examine and analyze Simple (human-crafted) Phishing URLs and AI-generated Phishing URLs through lexical features (a process of converting URLs into a sequence of characters) and word clouds (a visual representation of words which are frequently used in URLs) based analyses. We aim to analyze the differences in the formation and behaviour of Simple Phishing URLs and AI-generated Phishing URLs.

A. LEXICAL FEATURES-BASED ANALYSIS

To analyze lexical features-based behaviour and differences between Simple Phishing URLs and AI-generated Phishing URLs, we draw plots for behavioural analysis based on features' count against the number of URLs for both, i.e., Simple Phishing URLs and AI-generated Phishing URLs as shown in Figure 1 and Figure 2 respectively.

We thoroughly analyze both Figure 1 and Figure 2, and perform comparative analysis. Firstly, in Simple Phishing URLs, there are some URLs which consist of more than one colon(:), i.e., they are some URLs which contain ports in them, while AI-generated Phishing URLs never consist of ports. Secondly, in Simple Phishing URLs, there are few URLs in which more than one double forward slash(//) exist, i.e., besides segment part they incorporate double forward slashes in the path section of the URLs, while in AI-generated Phishing URLs double forward slashes are only use for the segment part. From plots, we can see that there is a high variance for dots(.) in Simple Phishing URLs. Sometimes they use few dots and sometimes upto 15 dots, i.e., sometimes Simple Phishing URLs incorporate other details in the URLs apart from domain name, SLDs(second-level domains) and TLDs(top-level domains). However, in AI-generated Phishing URLs, there is almost no variance. This shows that AI-generated Phishing URLs usually use dots to add more information besides domain name, SLDs and TLDs. Similarly, there is a high variance for virgule(/) feature in Simple

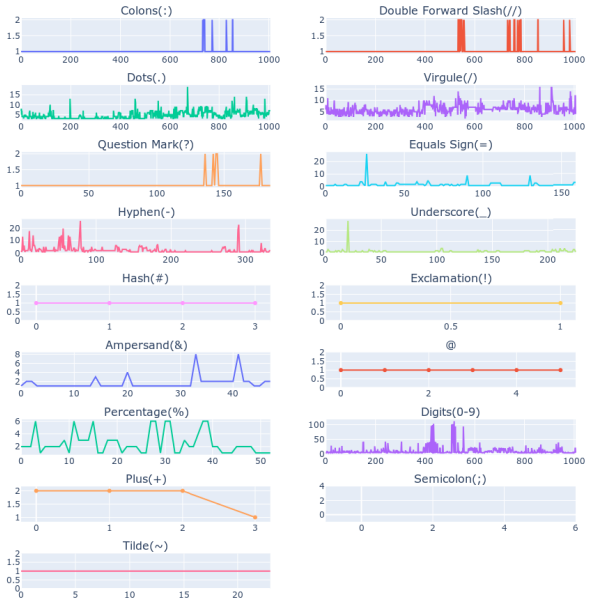

FIGURE 1. Behavioural analysis of selected lexical features from Simple Phishing URLs having the number of URLs on the x-axis and the features on the y-axis.

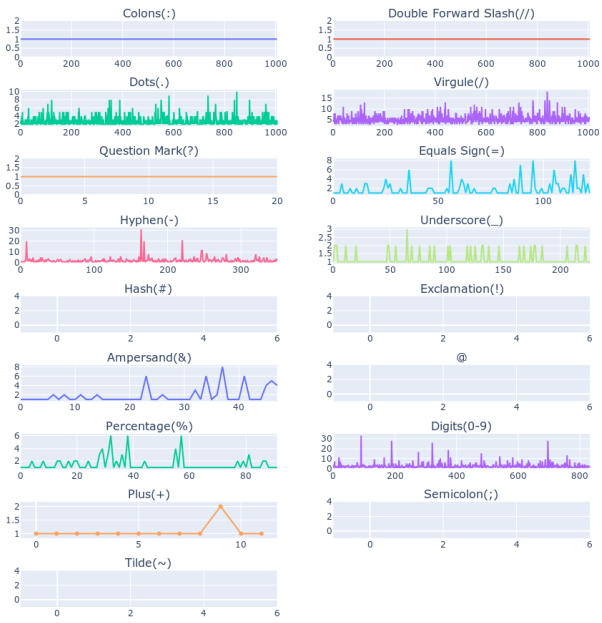

FIGURE 2. Behavioural analysis of selected lexical features from AI-generated Phishing URLs having the number of URLs on the x-axis and the features on the y-axis.

Phishing URLs. This means that sometimes Simple Phishing URLs incorporate different depths of hierarchical tree paths, while AI-generated Phishing URLs generally include longer depths of hierarchical tree paths. It can be seen that Simple Phishing URLs sometimes may use question mark(?) for other purposes. But in AI-generated Phishing URLs, they only count this feature for highlighting query part of the URLs. In plot, we can see that AI-generated Phishing

TABLE 1. Key comparative analysis between Simple Phishing URLs and AI-generated Phishing URLs based on specific lexical features.

S.No.	Features	Simple Phishing(SP) URLs	AI-generated Phishing(AIP) URLs
1	:	Some of the SP URLs contain ports in them	AIP URLs never have ports in them
2	//	Besides segment part, SP URLs incorporate double forward slashes in path of URLs	AIP URLs use double forward slashes only for segment part
3	.	Sometimes SP URLs incorporate other details to the URLs apart from domain name, SLDs and TLDs	AIP URLs usually use this feature to add more information besides domain name, SLDs and TLDs
4	/	Sometimes SP URLs incorporate different depths of hierarchical tree paths	AIP URLs generally include longer depths of hierarchical tree paths
5	?	Sometimes SP URLs may use it for other purposes	AIP URLs only use this feature for highlighting query part of the URLs
6	=	SP URLs rarely use this feature	AIP URLs use a lot of assignment of values, IDs, etc.
7	-	There is a great variation of words combinations in SP URLs	There is a subtle amount of variation of words combinations in AIP URLs
8	_	They have almost no variation in SP URLs	There is a great variation of words separations in AIP URLs
9	#!	There are very few SP URLs that contain both of them	AIP URLs don't use this feature
10	&	SP URLs rarely use this feature	AIP URLs mostly incorporate different numbers of queries in URLs
11	@	There are some SP URLs that incorporate user information part in them	AIP URLs don't use this feature
12	%	SP URLs frequently use URL HTML Encoding	AIP URLs use URL HTML Encoding less frequently
13	[0-9]	SP URLs use IDs, URL HTML Encodings and alphanumeric characters more frequently	AIP URLs use IDs, URL HTML Encodings and alphanumeric characters less frequently
14	+	SP URLs rarely use it but twice in a single URL	AIP URLs consume it sometimes, i.e., they include this feature once per URL
15	;	SP URLs don't use this feature	AIP URLs also don't use this feature
16	~	Some of the SP URLs include and specify home directory in them	AIP URLs never use this feature

URLs consider equals(=) sign in a reasonably good amount with great diversity, i.e., they use a lot of assignment of values, IDs, etc. However, in Simple Phishing URLs case, they rarely use equals sign. It can also be seen that there is a great variation of words combinations use in Simple Phishing URLs. On the contrary, there is a subtle amount of variation of words combinations in AI-generated Phishing URLs. As opposed to the hyphen(-) feature, there is a great variation of words separations in AI-generated Phishing URLs. Simple Phishing URLs, in contrast to this, have almost no variation for word separations.

Since hash(#) and exclamation(!) are used in combination for highlighting fragment part, therefore, we consider them together. In Simple Phishing URLs, there are very few URLs which contain both of them. On the other hand, there are none in AI-generated Phishing URLs. Plot for ampersand(&) feature shows that AI-generated Phishing URLs have a great diversity, i.e., they mostly incorporate different numbers of queries in URLs. On the other hand, Simple Phishing URLs consider ampersand feature rarely. In Simple Phishing URLs, we can see there are some URLs which contain @, i.e., there are some URLs that incorporate user information part in them. Conversely, there is none in AI-generated Phishing URLs category. For percentage(%) feature, Simple Phishing URLs use it in a fairly good quantity, whereas AI-generated Phishing URLs use it in a subtle amount. Hence, it shows that Simple Phishing URLs frequently use URL HTML Encoding as compared to the AI-generated Phishing URLs. On the same footing, it can be said that Simple Phishing URLs use digits([0-9]) quite frequently as compared to AI-generated Phishing URLs. This concludes that Simple Phishing URLs use IDs, URL HTML Encodings and alphanumeric characters more frequently than AI-generated Phishing URLs. For plus(+) sign, Simple Phishing URLs rarely use this feature. But whenever they consider this feature, they use it twice per URL. AI-generated Phishing URLs use plus sign sometimes as compared to Simple Phishing URLs. DeepPhish [1] usually include plus feature once per URL. Then for semicolon(;), none of them, i.e., Simple Phishing URLs and AI-generated Phishing URLs, consist of this feature. And finally, for tilde(~), there are some Simple Phishing URLs which include and specify home directory in them. However, AI-generated Phishing URLs do not use this feature.

Table 1 highlights the key comparative analysis of Figure 1 and Figure 2. The reason for performing the behavioural analysis using the specific lexical features is discussed in the subsequent section.

B. WORD CLOUDS-BASED ANALYSIS

To further investigate the behavior of AI-generated Phishing URLs, we employ Word Clouds approach. With Word Clouds

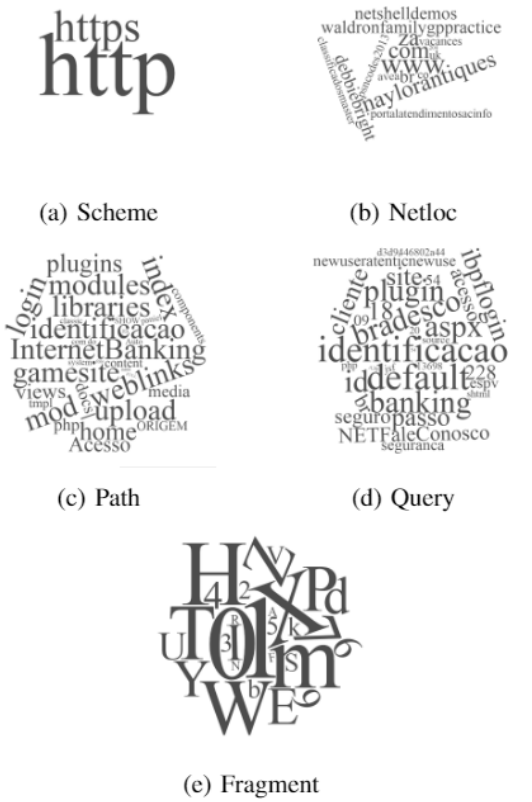

FIGURE 3. Frequent terms analysis from Simple Phishing URLs' Word Clouds.

approach, we are able to identify terms that are frequently used by DeepPhish [1] to generate phishing URLs.

From Figure 3, it can be observed that Simple Phishing URLs usually consist of five different parts, i.e., segment, netloc, path, query and fragment. From the Figure 3a, we can deduce that Simple Phishing URLs usually use **"http"** and **"https"** as communication protocols. For subdomain, Simple Phishing URLs use **"www"** services. In case of SLD, they have a length of three characters and use **"com"**. But they have no TLDs in them. For the given data, the frequently used term for domain name is **"naylorantiques"**. In Figure 3c, terms including **"php, html, index, upload, identificacao, accesso, weblinks, internetBanking, do, login, views etc."** are frequently used for path. These terms show that they have been used to fool users and to access users' information. On the other hand, in Figure 3d, mostly different values have been used with terms like **"id, passo, plugin, default, cliente"**. This means that attackers tried to redirect users to their pages or locations by assigning their locations' positions and IDs. And for fragment, we can see that Simple Phishing URLs consist of gibberish combinations of alphanumeric characters just to give a visual illusion to the Internet users about the phishing URLs as complete URLs.

Figure 4 shows that AI-generated Phishing URLs are made of four different parts, i.e. segment, netloc, path and query. Figure 4a depicts that **"http"** is generally used

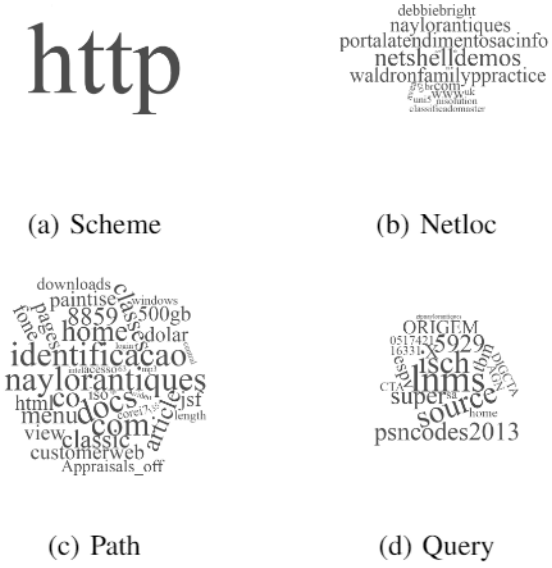

FIGURE 4. Frequent terms analysis from DeepPhish [1] URLs' Word Clouds.

as a communication protocol in AI-generated Phishing URLs. Similar to the Simple Phishing URLs, AI-generated Phishing URLs also use **"www"** services as a subdomain, **"com"** as an SLD and also have no TLDs in them. Given the dataset, we can see that AI-generated Phishing URLs used two different subdomains, i.e., **"naylorantiques"** and **"netshelldemos"**. In path, AI-generated Phishing URLs include words combinations like **"identificacao, naylorantiques, com, home, co, docs, menu, etc."** to hide their malicious sites or to deploy malicious codes by downloading different files on users' systems or to trick users to access users' credentials. For query part, DeepPhish [1] used combination of bogus terms like **"lnms, q, X, AGN, isch, sa, ORIGEM, tbm, espv, source, CTA, conta, etc."** with numbers to complicate query part. Hence by over-complicating queries, users are unable to understand things correctly and can be easily deceived by attackers.

Therefore, based on our exploratory data analysis, we can conclude that AI-generated Phishing URLs are usually made up of features that are somehow similar to the Normal URLs. Due to this, AI-generated Phishing URLs generally tend to look similar to that of Normal URLs. Thus, most of the time AI-generated Phishing URLs easily bypass simple phishing detection systems.

Therefore, this pose a need for a detection system which is able to even detect AI-generated Phishing URLs efficiently and effectively.

III. PHISHING URLs' DESIGN APPROACHES

Since URLs are the first thing that can be used to analyze and classify any website as phishing or normal. Phishing URLs always have some distinctive features. Those features can be incorporated in the phishing URLs through the following main approaches.

1) HIDDEN LINKS

One way of persuading a victim to click on the phishing link is through Hidden Links. Hidden Link is a type of technique through which attackers hide the phishing URLs through:

1) Using some keywords, e.g., ''CLICK HERE'', ''DOWNLOAD'', ''SUBSCRIBE'', etc.
2) Replacing with IP addresses
3) Appending other domain names (mostly different brand names)

This technique helps attackers to easily deceive a victim and launch a phishing attack without displaying a phishing URL.

2) COMBOSQUATTING

It is also commonly known as ''Cybersquatting'' or ''Domain squatting''. In this technique, attackers either register or use the domain name for phishing purposes.

Combosquatting can be carried out in either of the following ways:

1) Correctly spelled domain name but appending a string to it that appears legitimate and anyone can register, e.g., microsoft-login.com
2) Omitting a period, also known as ''Doppelganger'' domain. Doppelganger is a type of domain that has identical spelling to that of a legitimate FQDN(Fully Qualified Domain Name) but has missing dots between subdomain and domain, e.g., enwikipedia.org instead of en.wikipedia.org
3) Adding an extra period, e.g., air.france.com instead of airfrance.com

3) TINY URLs

URL shortening is an approach that helps in shortening the expanded or lengthy URLs, hence resulting in tiny URLs. URL shortening is desirable due to various reasons which include but not limited to make URLs aesthetically pleasing, hide underlying confidential addresses, and provide an ease to remember URLs.

4) TYPOSQUATTING

It is also known as ''URL Hijacking'', a ''Sting Site'', or a ''Fake URL''. It is a form of cybersquatting. It solely relies on mistakes, in terms of typos, either made by Internet users while entering the websites in the web browsers or based on typographical errors which are hard to notice while quick reading. Hence, any typo error may lead to a phishing page. This also includes brandjacking.

Typosquatting is usually carried out in one of the five following ways:

1) Simply misspelled, e.g., applle.com instead of apple.com
2) Typo based misspelled, e.g., appel.com instead of apple.com
3) Different domain name, e.g., apples.com instead apple.com
4) Different TLD, e.g., apple.org instead of apple.com
5) ccTLD(Country Code top-level domain), e.g., apple.cm, apple.co etc. instead of apple.com

We take into account all these four types of techniques. Our proposed solution, methodology, and approaches are capable enough to deal with all of the above mentioned types of phishing URLs techniques.

IV. PROPOSED SOLUTION: PhishHaven

We propose PhishHaven, a novel phishing URL detection system. It works as a browser plugin as shown in Figure 5. The main novelty of PhishHaven lies in its detection, i.e., it is especially crafted to detect AI-generated Phishing URLs. Furthermore, PhishHaven is capable to deal with tiny URLs with our URL Hit approach. In addition to this, it is unique in terms of its parallel execution of ensemble machine learning models along with unbiased voting-based classification.

A. DESIGN PRELIMINARIES

To understand how our proposed system works, we need to understand the following preliminaries.

1) URL HIT

With the motive to obtain sensitive information such as pass codes, personal credentials etc., phish attackers generally use tiny URL approach. Therefore, to detect tiny URLs efficiently, it is important to understand how tiny URLs work.

The URL shortening works because of browsers' redirect functionality. When a user clicks or types a tiny (shortened) URL in the browser, the browser sends HTTP request to the server directing it to fetch the requested page, after which the server then sends either of the following redirect requests [29]:

1) 301: Moved Permanently
2) 302: Found
3) 303: See Other
4) 304: Not Modified
5) 305: Use Proxy
6) 307: Temporary Redirect

There is a community of adversaries who took the leverage of URL shortening approach to fulfill their adversarial goals. Tiny URL is also one of the characteristics of phishing URLs. Through tiny URLs, attackers can easily hide paths of malicious pages or deploy a malicious piece of code. Thus, detection of tiny URLs is also essential. But the characteristics of tiny URLs make it difficult for the detection systems to detect tiny URLs.

Although, in [21] and [23], the authors have a rich list of lexical features and employed various detection approaches; their models were unable to detect tiny URLs.

With the main aim to detect AI-generated Phishing URLs, we also introduce a URL Hit approach to efficiently deal with tiny URLs. It is incorporated into our detection system in a way that whenever a user clicks a URL, let's say (I_{URL}), the URL firstly redirects toward our plugin, PhishHaven. The URL (I_{URL}) is then further hit by our plugin. We then fetch the response of the respective hit URL (I_{URL}) in terms of

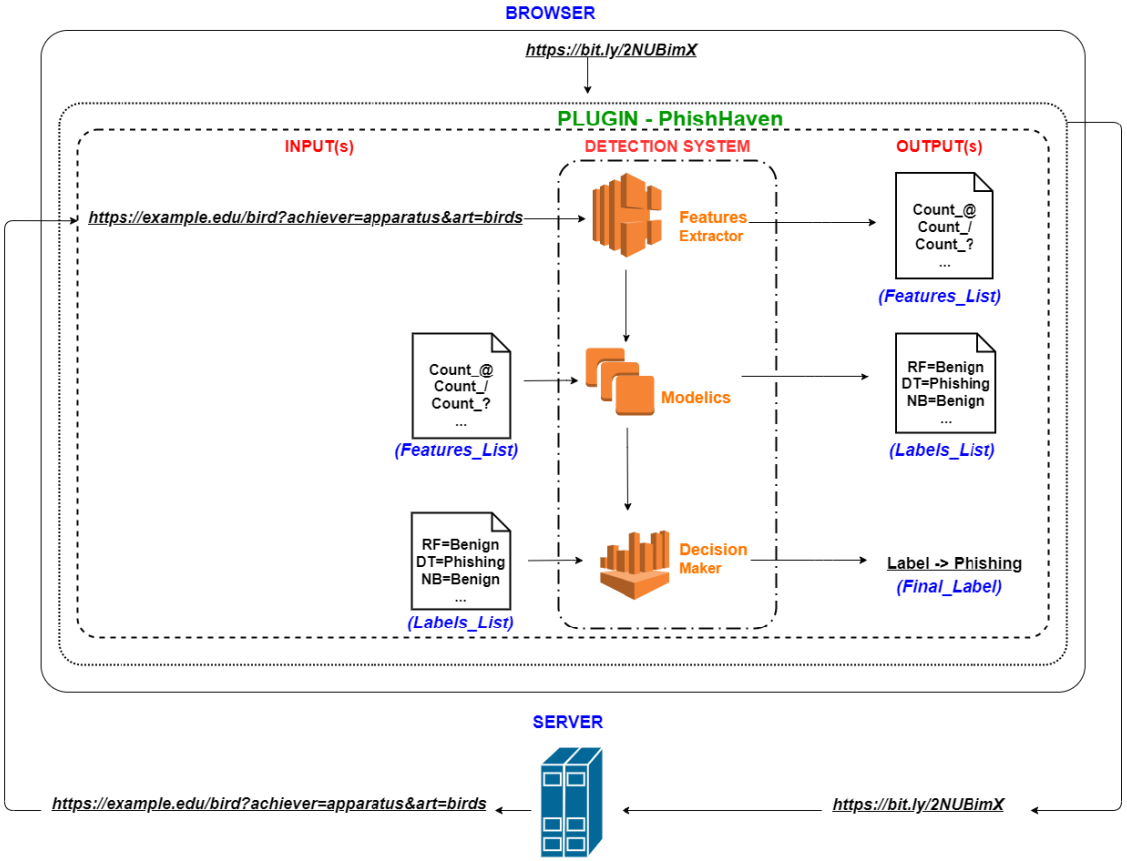

FIGURE 5. The architecture of PhishHaven. PhishHaven, a browser plugin, takes a URL and employs URL Hit approach to extract an extended URL from the server. Then using the Features Extractor subcomponent, PhishHaven extracts selected lexical features from an extended URL. Next PhishHaven executes the Modelics subcomponent to generate a list of labels, i.e., phishing or normal. Finally, PhishHaven uses Decision Maker subcomponent to assign the final label to the initial entered URL.

an extended URL, i.e., it returns an actual URL, let's say (A_{URL}), after which the very first component of our detection system, i.e., Features Extractor extracts features from URL (A_{URL}).

2) FEATURES EXTRACTOR

In order to detect AI-generated Phishing URLs, we employ an adversarial learning approach. Through this approach, we extract various features that are found specifically in AI-generated Phishing URLs. Thus, this approach enables PhishHaven to detect AI-generated Phishing URLs more accurately.

a: FEATURES SELECTION

There can be a wide variety of lexical features for classifying URLs. In our study, we specifically focus on AI-generated Phishing URLs. Therefore, we analyzed how similar or distinguishing are the AI-generated Phishing URLs to that of Simple Phishing and Normal URLs. In this study, we only analyze DeepPhish [1] generated URLs which is, to the best of our knowledge, the only AI model designed for generating phishing URLs.

In the future, there may be many other AI models designed for generating phishing URLs. Therefore, we study how Google deals with different URLs which are designed for different purposes. With this technique, our detection system, PhishHaven, is able to train on various yet most significant lexical features.

The two main categories of special characters used to create a full-fledged URL are:

1) Reserved Characters:
   These include dollar sign ($), ampersand (&), plus (+), common (,), virgule (/), colon (:), semi-colon (;), equals sign (=), question mark (?), and at (@) symbol. All of them have various purposes and significant meanings when used in URLs.
2) Unreserved Characters:
   These include space ( ), quotation marks (‘ ’), less than (<), greater than (>), hash (#), percent (%), left curly brace ({), right curly brace (}), pipe (|), backslash (\), caret (^), tilde (∼), left square bracket ([), right square bracket (]), and grave accent (‘). They are also known as ‘‘Unsafe Characters’’. These characters if not encoded properly within the URLs can be easily misunderstood for various reasons.

Based on these two major categories, we select the list of lexical features deliberately as mentioned in Table 2 along with their reasons.

TABLE 2. List of selected lexical features along with their characteristics and reasons for their selection.

S.No.	Features	Characteristics	Reasons
1	:	This feature separates port in the URL.	It is used to identify whether a URL has port or not.
2	//	This feature indicates segment in the URL.	It is used to identify segment part.
3	.	This feature determines the relative references within path hierarchy, domain name, SLD, and TLD.	It is used to extract these information.
4	/	This feature helps in creating hierarchy within path.	It is used to compute the path's length in the URL.
5	?	It determines the query section in the URL.	It is used to identify the query section of the URL, if any.
6	=	This feature is used as an assignment operator in the URL.	It is used to identify the assignment of values in URL.
7	-	Google treats this feature as a word separator.	It is used to count the number of words separators.
8	_	Unlike hyphen, Google treats this feature as a word joiner.	It is used to count the number of words joiners.
9	#	This feature specifies the fragment part in the URL.	It is used to extract the fragment part of the URL.
10	!	Fragment is never being sent to the remote Web server and is entertained at local Web browser level. Hence, this makes two URLs pointing at different locations of the same page similar for search engines. In order to resolve this problem, Google provides a syntax known as "hash bang" , i.e., hash followed by exclamation mark (#!), for URLs to interpret two URLs differently (if they have different fragments).	It is used to extract different fragments from the URL, if any.
11	&	This feature is used as a delimiter in query part.	It is used to count the number of queries in the URL.
12	@	This feature individuates the user information.	It is used to extract and analyze whether any user information exists in the URL or not.
13	%	This feature is used in combination with alphanumeric characters for **URL HTML Encoding** [30].	It is used to identify HTML Encoded characters in the URL.
14	[0-9]	This feature is also usually used in **URL HTML Encoding**.	It is used to identify **URL HTML Encoding**, existence of IDs (i.e. Session IDs, Affiliate IDs, Tracking IDs, Referrer IDs) and Time stamps.
15	+	In URL encoding, spaces are encoded to plus signs.	It is used to count the number of spaces in the URL.
16	;	It is used as a substitute of ampersand	It is used for counting the number of queries in the URL.
17	∼	This feature shows reference to home directory, i.e., cpanel location which is generally relative to the home directory.	It is used to identify whether the URL has any such reference or not.

b: URL HTML ENCODING, A LEXICAL FEATURE

URL HTML Encoding, also known as character encoding, is essential as it is required to process non-ASCII characters of the URL. Also, to improve the page-loading time, especially on slow connections, HTML Encoding is indicated in the first 1024 bytes of the document.

Therefore, an attacker can play around with character encoding to hide malicious information embedded in the URL. Hence, we consider URL HTML Encoding as an additional lexical feature for phishing URL detection.

Since Features Extractor subcomponent is responsible for extracting lexical features from an extended URL; we develop an approach for this component. We extract features into two parts.

1) **Overall count**
In this part, we first take the URL as a whole. Then we extract all the selected lexical features mentioned in Table 2 one by one from the whole URL.

2) **Individual count**
On the other hand, in this part, we first divided URL into the following five major components, i.e, segment, netloc, path, query and fragment.

a) **Segment** is also known as "scheme". It determines the type of protocol used for accessing the resources on the Internet, e.g., https, http, ftp, etc.
b) **Netloc** is also known as "hostname". It determines the registered name, an IP address and user information. It is further divided into different subcomponents which are subdomain, domain name, top-level domain, second-level domain and port.
   i) **subdomain** determines the type of service used for accessing the resources, e.g., www, video, etc.
   ii) **domain name** determines registered entity, e.g., google.

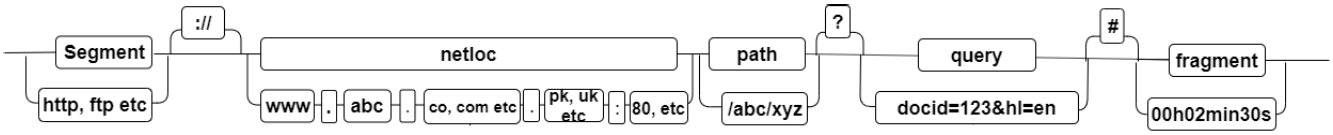

FIGURE 6. Lexical features extraction approach from the five components of a URL.

iii) **top-level domain or TLD** determines country code, e.g., pk, uk, etc.
iv) **second-level domain or SLD** determines the type of a registered entity, e.g., org, edu, com, etc.
v) **port** determines which port is used for communication between client and server.

c) **Path** determines the location of a file on the web server, e.g., directory/folder/file.
d) **Query** determines which type of request(s) are made by a user to the web server, e.g., docid=−12548987, etc.
e) **Fragment** is also known as ''anchor''. It determines internal page or internal section references, e.g., category=blue, etc.

After dividing a URL into its respective five components, we then extract features from features list in Table 2 from each component, as shown in Figure 6.

3) MODELICS

Modern operating systems are capable enough to execute multiple tasks concurrently using multi-threading approach.

Therefore, we leverage the multi-threading to detect AI-generated Phishing URLs efficiently in real-time. We design this subcomponent in a way that it acts as a single process composed of multiple threads. These multiple threads are all the machine learning models running (learning or predicting) simultaneously, but independent of each other.

Each model (thread) takes extracted features from previous subcomponent, Features Extractor, as an input and then sends its respective predicted result individually and independently to the Decision Maker, the next subcomponent of our detection system.

4) DECISION MAKER

To avoid simple majority-based voting concept which suffers from a limitation of having equal number of votes for each value or state; we borrow the voting concept from Fault-Tolerant mechanism which is used by distributed systems to achieve a necessary agreement on a single value or single state by a majority of $\frac{2}{3}$ or 67% and design our mechanism named as Voting.

Our Voting mechanism takes simultaneously generated prediction results (individual and independent) of each machine learning model and then makes a final decision about the class of a URL based on 67% of classifiers classifying for either of the class, i.e., phishing or normal.

Algorithm 1 Algorithm for URL Hit

Output: Expanded_URL
Set variable: Tiny_URL
Expanded_URL = requests.get(Tiny_URLs[index])

In a nutshell, our PhishHaven takes a URL from a browser. It then employs our URL Hit approach on it and fetches an extended URL. Thereafter, Features Extractor extracts features from an extended URL, after which the extracted features are sent to the Modelics subcomponent. And at last, the Decision Maker subcomponent takes all the outputs from the Modelics subcomponent, decides and assigns a Final Label, i.e., phishing or normal to the initially entered URL.

B. METHODOLOGY

With the aim to design an efficient real-time phishing URLs detection system, we design a PhishHaven which detects and classifies a URL using four subcomponents. First subcomponent, URL Hit which extracts extended URLs from tiny URLs. The second subcomponent is Features Extractor which extracts selected lexical features from the extended URLs. Then the third subcomponent, Modelics, which executes ensemble-based machine learning models in parallel and collects the classification results. And lastly, Decision Maker subcomponent which assigns a final class, i.e., phishing or normal to the URL.

1) URL HIT

When a user enters a URL, a URL is first redirected towards our detection system. Thereafter, the URL is further hit with a request of a response in terms of URL by our detection system. The requested respective response can be either an expanded URL in case of tiny URL or the same URL in case of expanded URL as shown in Algorithm 1. The requested respective response is then passed to the next subcomponent, i.e., Features Extractor.

2) FEATURES EXTRACTOR

To extract features from extended URL(s), this subcomponent first extracts features from URL(s) as a whole using Algorithm 3. Then it divides the expanded URLs into their respective five parts, i.e., segment, netloc, path, query and fragment as shown in Algorithm 2. After this, the respective parts of URL(s) also undergo the process of features extraction as shown in Algorithm 3. We use regular expressions to extract different types of features after which the extracted features become the output of this subcomponent.

Algorithm 2 Algorithm for Components Extraction From URLs

```
Input: Expanded_URLs
Output: Segment_List
Netloc_List
Path_List
Query_List
Fragment_List
Set variable: index = 0
for url in Expanded_URLs do
    parser = urlparse(url)
    Segment_List.insert(index, parser.scheme)
    Netloc_List.insert(index, parser.netloc)
    Path_List.insert(index, parser.path)
    Query_List.insert(index, parser.query)
    Fragment_List.insert(index, parser.fragment)
    index = index + 1
end for
```

Algorithm 3 Algorithm for Features Extraction From URLs and Their Components

```
Input: Expanded_URLs, Segment_List, Netloc_List,
Path_List, Query_List, Fragment_List
Output: Features_List
for index in No._Features do
    Features_List.insert(index, extracted_features)
end for
```

3) MODELICS

For this subcomponent, different machine learning models are set in parallel as individual and independent threads. A set of extracted features from the previous subcomponent, Features Extractor, becomes an input to this subcomponent.

This subcomponent, Modelics, consists of ten machine learning models. We categorize machine learning models according to our case into three categories, i.e., boosting-based approach, non-learning-based approach and learning-based approach because we want to introduce variance in decision making and feature selection processes.

- **Boosting-based approach**
  Classifiers in this category employ voting-based decision and focus on making weak learners strong. In this category, we consider AdaBoost and Gradient Boosting classifiers for the following reasons:
  - **AdaBoost Classifier** alters the distribution of the samples in the training dataset to increase the weights of the training samples which are difficult to classify. Moreover, it focuses on making weak learners strong by assigning weights to the weak learners based on data misclassifications. Also, it makes a final prediction based on the majority vote by taking the weak learner's predictions which are weighted by their individual accuracy. [31], [32]
  - **Gradient Boosting Classifier** minimizes the overall error of strong learners through gradient optimization process on each weak learner. It also minimizes the loss function of the strong learner in order to focus on misclassified samples in the training dataset [33].
- **Non-learning-based approach**
  Classifiers in this category are less prone to over-fitting. For this category, Decision Trees, Random Forest, Extra Tree classifier, Bagging classifier and K-Nearest Neighbour are considered along with the following reasonings:
  - **Decision Trees** considers all the features in the entire dataset at a time while constructing a model. It has a high variance [34].
  - **Random Forest** considers random features from the entire dataset at a time while constructing a model. The selection of features is carried out on the basis of best split and with replacement(bootstrapping). It has a medium variance [35].
  - **Extra Tree Classifier** also considers random features from the entire dataset at a time while constructing a model. Whereas, the selection of features is carried out on the basis of random split and without replacement. It has a low variance [36].
  - **Bagging Classifier** makes multiple groups comprised of different features from the entire dataset. It considers a random combination of features with the groups. The goal of this classifier is to decrease the variance [32].
  - **K-Nearest Neighbour** generates clusters based on features and samples in the training dataset. It then assigns a cluster to the new sample based on the distance between the new sample and clusters [37].
- **Learning-based approach**
  This category's classifiers perform classification through optimization and drawing decision boundaries. We choose three classifiers from this category, i.e., Logistic Regression, Support Vector Machines and Neural Networks because of the following reasons:
  - **Logistic Regression** makes a binary classification. It is a probabilistic approach and assigns samples to the class based on class probability [34], [38].
  - **Support Vector Machines** draws a decision boundary based on features from the dataset in the hyperplane. It is a deterministic approach and, requires optimization and regularization [39].
  - **Neural Networks** learns the distribution of the samples in the training dataset and performs classification based on optimized weights and biases. It also performs multi-layer optimization by itself [38], [40].

Each machine learning model takes a set of features and performs learning (in learning case) and prediction (in prediction case). Then every machine learning model predicts the class label and sends its prediction result to the next

subcomponent. Each model produces output individually and independent of each other.

We can express this subcomponent mathematically as,

$$E(X) = \begin{cases} AB(X) = sign(\Sigma_{i=1}^{T} \alpha_i h_i(X)) \\ \overline{B}(X) = sign(\Sigma_{i=1}^{T} sign(f_i(X))) \\ DT(X) = \Sigma_{i=1}^{C} - f_i(1 - f_i) \\ XT(X) = \Sigma_{i=1}^{X} - f_i(1 - f_i) \\ GB_m(X) = GB_{m-1}(X) + \Sigma_{i=1}^{T} \alpha'_{im} 1_{R_{im}}(X) \\ KNN(X) = P(y = j|X = x) = (\frac{1}{k} \Sigma_{i \in A} I(y^i = j)) \\ LR(X) = WX + B \\ NN(X) = f(b + \Sigma_{i=1}^{n} X_i w_i) \\ \hat{RF}(X) = \frac{1}{B'} \Sigma_{i=1}^{B'} RF_i(X') \\ SVM(X) = \begin{cases} +1, & if\ w.X + b' \geq 0 \\ -1, & if\ w.X + b' < 0 \end{cases} \end{cases} \tag{1}$$

Here, E is multi-threading of Ensemble Machine Learning Models where every model is an individual and independent thread. AB is an AdaBoost Classifier having X as a set of extracted features along with α, the weights for the classifiers and T, the number of classifiers. $\overline{B}$ is a Bagging Classifier consisting of the frequency of the label (f_i), a set of extracted features (X) and the number of classifiers (T). Next is the Decision Tree (DT) made up of C number of unique labels and frequency of the label (f_i), whereas XT, an Extra Tree Classifier consists of X, a set of extracted features and frequency of the label (f_i). For Gradient Boosting Classifier (GB), it has a set of extracted features (X), m number of iterations, T number of classifiers, step size of α' and pseudo residual (R). In KNN, K-Nearest Neighbour, it has probability P, feature (x), set of extracted features (X), k number of neighbours, y number of clusters, set of points close to x (A) and new sample (j). The Logistic Regression (LR) model uses the co-efficients of extracted features (W), a set of extracted features (X) and a slope (B). For NN, Neural Networks, parameters include bias (b), n number of inputs from the incoming layer, input to neuron (X_i) and weights (w_i). The parameters Random Forest ($\hat{RF}$) utilizes include B' the times of bagging, regression tree (RF_i) and (X') as a root feature of (RF_i). And for SVM, Support Vector Machines, X is a set of extracted features, w is the representation of hyperplane in terms of line and b' is the representation of slope in terms of line.

The Algorithm 4 also shows the working of Modelics subcomponent.

4) DECISION MAKER

Finally, in this subcomponent, a set of prediction results from the previous subcomponent, Modelics, becomes an input. This subcomponent is responsible for collecting all the results and deciding the final class of an initially input URL in the Features Extractor subcomponent.

Algorithm 4 Algorithm for Modelics

Input: Features_List
Output: Labels_List
Set variable: index = 0
Set: threads_List = list( )
Set of ML models: Models = $\{M_1, \ldots, M_N\}$
(Upto N-Procedures)
procedure MODELICS(X_train, y_train, X_test, que)
model = Classifier().fit(X_train, y_train)
prediction = model.predict(X_test)
queue.put(prediction)
end procedure
for M in Models **do**
M = threading.Thread(target = MODELICS, args = (X_train, y_train, X_test, que)
M.start()
threads_List.append(M)
end for
for t in threads_List **do**
Labels_List.insert(index, queue,get())
t.join()
index = index + 1
end for

Decision Maker first collects all the results and then makes a count of the total number of predicted results which are labelled as either phishing or normal. Based on the total count, it then checks for the Final Label as following:

$$FinalLabel = \begin{cases} Phishing, & \text{if } NoPL > NoNL \\ Normal, & \text{if } NoPL < NoNL \end{cases} \tag{2}$$

where,

NoPL is the Number of Phishing Labels

NoNL is the Number of Normal Labels.

Since phishing attacks result in severe damages (e.g., theft of highly confidential information). Therefore, to refrain every possibility of phishing attack; we consider 67% of the classifiers voting for a single class at a time.

Hence, in a case where both classes (i.e. phishing and normal) have equal number of votes, the Modelics operation is re-performed. On the other hand, in a scenario where 67% of the classifiers classify a URL as phishing, it is classified as phishing, i.e., Case-I. Otherwise as normal, i.e., Case-II. The Algorithm 5 shows the working of the Decision Maker subcomponent.

C. TIME COMPLEXITY ANALYSIS

We perform the time complexity analysis of our proposed five algorithms, as shown in Table 3. Firstly, for URL Hit algorithm, we consider one URL at a time. Hence URL Hit consumes a constant time, i.e. $O(1)$. In Algorithm 2, PhishHaven extracts five components from each expanded URL consuming a constant time $O(1)$.

Algorithm 5 Algorithm for Decision Maker

```
Input: Labels_List
Output: Final_Label
Set variable: count_Phishing = 0
Set variable: count_Normal = 0
for index in Labels_List do
  if Labels_List[index] == Phishing then
    count_Phishing = count_Phishing + 1
  else
    count_Normal = count_Normal + 1
  end if
end for
if count_Phishing > count_Normal then
  Final_Label = "Phishing"
else if count_Phishing < count_Normal then
  Final_Label = "Normal"
else
  GO TO PROCEDURE MODELICS()
end if
```

TABLE 3. Time complexity analysis of algorithms.

Algorithms	Time Complexity
URL Hit	O(1)
Components Extraction from URLs	O(N)
Features Extraction from URLs and their Components	O(1)
Modelics	$O(M+N^3)$
Decision Maker	$O(M+N^3)$

Therefore, in the worst-case scenario, PhishHaven will extract five components from N expanded URLs leading to time complexity of $O(N)$, after which PhishHaven employs Algorithm 3 for extracting seventeen lexical features from both URLs as a whole and their components, thus consumes a constant time of $O(1)$. Next for Modelics algorithm, PhishHaven runs machine learning algorithms in parallel but independent of each other through multi-threading. We choose ten machine learning algorithms as mentioned and discussed under the Modelics subcomponent. Machine learning algorithms which consume the longest computation time are Neural Networks and Support Vector Machines, i.e. M(represented as a total time complexity including k time for training, l number of epochs, i number of nodes in the first layer and j number of nodes in the second layer of the model i.e. $k * l * i * j$) and N^3, respectively. Therefore, PhishHaven needs to wait for Neural Networks and Support Vector Machines to complete their computation, thus resulting in overall time complexity for Algorithm 4 as $O(M + N^3)$. Finally, for Decision Maker algorithm, PhishHaven consumes a constant time of $O(1)$ to classify a URL in the best and average case. But in the worst-case scenario PhishHaven requires to re-compute Algorithm 4, thus requires a time complexity of $O(M + N^3)$.

TABLE 4. Datasets: Types, sizes and sources.

Types	Sizes	Sources
Normal URLs	50,000	Alexa [42]
Simple Phishing URLs	50,000	Phishtank [43]
AI-generated Phishing URLs	50,000	DeepPhish [1]

V. EXPERIMENT

A. SIMULATION ENVIRONMENT

To perform our experiments, we choose the LINUX Ubuntu 16.04 environment. The detection system is developed with Python version 3.5.4. For features extraction and data analysis, we use the Python libraries including nltk, re, numpy, matplotlib, wordcloud, counter, plotly, urlparse and dnstwist[1] [41]. And to design the Modelics subcomponent, we use pandas, sklearn, threading and seaborn libraries.

B. DATA PREPARATION

To achieve our main goal, i.e., to detect AI-generated Phishing URLs, we used DeepPhish [1], an AI-based model developed for generating phishing URLs. From DeepPhish [1], we generated 50,000 AI-generated phishing URLs. For Normal URLs, we took 50,000 URLs from Alexa. Also, to test our PhishHaven against Simple Phishing URLs; we took 50,000 Simple Phishing URLs from PhishTank.

To include and prepare our datasets according to recent phishing cases and current URLs' format, we downloaded datasets by September 4th, 2019 from both PhishTank and Alexa.

To have a fair evaluation of our PhishHaven against AI-generated Phishing URLs, we consider a 5:5 split ratio for training and testing datasets. To prepare our experimental datasets, we first combined AI-generated Phishing URLs and Normal URLs and shuffled them randomly. Then we applied Hold-out method to split the dataset into training and testing datasets. Due to the limitation of variation in the patterns of the features presented in the URL's generated from DeepPhish [1], applying K-fold validation for evaluating our models' efficiency can result in a biased generalization. Hence, we choose to perform Hold-out method to introduce variation in the training set while focusing on limiting the overfitting or over-generalization. For Simple Phishing URLs, we performed the same evaluation procedure as that of AI-generated Phishing URLs.

C. EVALUATION METRICS

In the field of cybersecurity, security comes first. Keeping this aspect in mind, we first need to choose the evaluation metric, that is, which type of performance measure(s) we want to increase or decrease or we are more considerate about.

Since phishing attacks can cause severe damages and harm to the end-users, we decide to choose **Sensitivity** (also known

[1]Dnstwist is a domain name permutation search tool, which can detect typoquatting, bitsquatting, and fraudulent websites that share similar looking domain-names

TABLE 5. Confusion matrix.

	Predicted: False	Predicted: True
Actual: False	TN	FP
Actual: True	FN	TP

as **"True Positive Rate"** or **"Hit Rate"**) as our one of the main evaluation metrics. TPR is a probability of correctly detecting the H_0 (null hypothesis).

Here we have considered

$$H_0 : Entered\ URL\ is\ phishing$$

Moreover, we also consider **Fall-Out** (also known as **"False Positive Rate"**). FPR is a probability of falsely detecting the H_0 (null hypothesis).

To avoid any hindrance faced by users in availing services, we also evaluate our system through **Specificity** (also known as **"True Negative Rate"**). TNR is a probability of correctly rejecting the H_0 (null hypothesis).

where,

TP = correctly identified

TN = correctly rejected

FP = incorrectly identified

FN = incorrectly rejected

$$True\ Positive\ Rate = \frac{TP}{TP + FN}$$
$$False\ Positive\ Rate = \frac{FP}{FP + TN}$$
$$True\ Negative\ Rate = \frac{TN}{TN + FP}$$

To evaluate how accurate our PhishHaven works, we also choose **"Accuracy"**. The Accuracy is defined as

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}.$$

Our main objective behind the selection of TPR, FPR and TNR is to always prevent a user from any phishing attacker's attacks irrespective of any level of loss or damage while making sure that there are significantly less misclassifications regarding Normal URLs.

Further to evaluate the performance of our selected machine learning models, we choose to include are Precision, Recall and F1-measure as;

$$Precision = \frac{TP}{TP + FP}$$
$$Recall = \frac{TP}{TP + FN}$$
$$F1\text{-measure} = 2\frac{Precision * Recall}{Precision + Recall}$$

D. PhishHaven ANALYSIS

We evaluate PhishHaven using theoretical analysis as well as experimental analysis. Through theoretical analysis, we examine the logical structure of concepts and statements of PhishHaven. Through experimental analysis, we experimentally prove the logical structures of concepts and statements of PhishHaven.

1) THEORETICAL ANALYSIS

The purpose of conducting the theoretical analysis is to demonstrate that there is a valid argument in favour of our proposed hypothesis, i.e.,

Phishhaven Can Detect Ai-Generated Phishing URLs

To prove our hypothesis, we propose two propositions: Proposition 1, to prove that PhishHaven is capable of detecting every tiny URL; and Proposition 2, to prove that our selected lexical features are invariant, hence, leading to a proof that PhishHaven can detect AI-generated Phishing URLs.

PROPOSITION 1: URL HIT WORKS FOR EVERY TINY URL

Since PhishHaven is a plugin which means that PhishHaven works as a middle party between users and servers, all the entered tiny URLs are first redirected to PhishHaven. PhishHaven then employs the URL Hit approach, i.e., further sends the tiny URL(s) to the server. Thereafter, in response of the sent tiny URL(s), PhishHaven receives an extended URL(s), and then the features extraction procedure is performed on the respective extended URL(s).

Hence, we can say that PhishHaven is one to one correspondence (or bijection) function, i.e.:

$$PhishHaven(I_{URL}) = A_{URL}\ \text{implies}\ I_{URL} \rightarrow A_{URL}$$

where,

I_{URL} = initial URL (tiny URL),

A_{URL} = response URL from server (always be extended).

Furthermore, our proposed URL Hit approach is comprised of three main postulates.

- **Postulate 1: Default behaviour**
  We set URL Hit as the default behavior of our detection system. With default behavior, we mean irrespective of the URL form, i.e., either extended or shortened; a URL always hit by our detection system with a request of a response in terms of an actual URL. By setting up this approach, we are able to cater the case of tiny URLs as the classification is directly applicable on an actual URL(s) (A_{URL}).
- **Postulate 2: Independent of prior knowledge**
  Since we leverage the browser's redirection property, our URL Hit approach is independent of any URL shortening software, e.g., bitly, TinyURL, Polr, etc. URL Hit works for every URL including those URLs which are shortened by any algorithm, approach or methodology. Also, with our approach, any prior information either regarding URL or shortening software or shortening algorithm is not required.
- **Postulate 3: Consistent**
  Our proposed approach is consistent, i.e., on a given input, we always get the same output. The results from

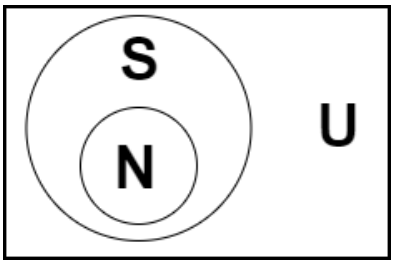

FIGURE 7. Case-1.

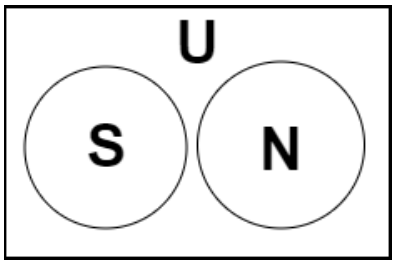

FIGURE 8. Case-2a.

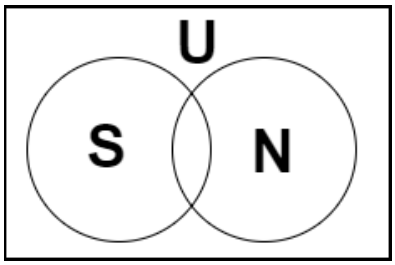

FIGURE 9. Case-2b.

our approach are free from any randomness and measurement errors.
In other words, the probability of getting the same extended URL, i.e., (A_{URL}) in response of a URL, i.e., (I_{URL}) is always 1.
Mathematically this can be defined as:
Since the relationship is $I_{URL} \rightarrow A_{URL}$, therefore

$$Pr[URL\_Hit(I_{URL}) = A_{URL}] = 1.$$

Our assumption for this postulate is that the server is always active.

Therefore, this proves our Proposition 1 that our URL Hit approach works for every tiny URL.

PROPOSITION 2: SELECTED FEATURES ARE INVARIANT

PhishHaven extracts and analyzes a set of those lexical features which are invariant, i.e., a feature from that set must be a part of a full-fledged URL. Let U be the superset of lexical features, S be the set of selected lexical features, M be the set of lexical features extracted from the DeepPhish [1] URLs and N be the set of lexical features extracted from the future AI-generated Phishing URLs.

This proposition consists of two main cases:

- **Case 1:** $N \subset S$
  Since M and N are the subsets of S, i.e.,

  $$M \subset S \; and \; N \subset S,$$

  we can say that all the elements in N must be in S, i.e.,

  $$N = \{\forall e \in N : e \in S\}.$$

  Hence, this implies that

  $$Pr[\forall e \in N : e \in S] = 1.$$

- **Case 2:** $N \not\subset S$
  This case is further divided into two cases.
  1) **Case 2a:** $(N \cap S) = \mathbf{0}$
     Since S is comprised of a set of significant lexical features, and the probability of N not in the proper subset of S is negligible, we write it as

     $$Pr[N \subset U : \nexists e \in S] = \epsilon.$$

  2) **Case 2b:** $(N \cap S) \subset S$
     Since S contains essential lexical features, which are necessary to construct a full-fledged URL, it is evident that N must consist of at least one of the lexical features from the set S, i.e,

     $$Pr[\exists e \in N : e \in S] = 1.$$

     In other words, we can express it as

     $$N = \{\{x\}, \{y\} : \{x\} \in U, \{y\} \in S\}.$$

Based on three cases, we can prove that our selected features are invariant. Hence, it can be said that a feature from a set of our selected lexical features must be a part of every full-fledged URL.

Through Proposition 1 and 2, we proved our proposed hypothesis. Therefore, we can say that PhishHaven can detect future AI-generated phishing URLs with 100% accuracy based on URL Hit and our selected lexical features.

2) EXPERIMENTAL ANALYSIS

We perform experimental analysis of PhishHaven to support our theoretical analysis. In our experimental analysis, we perform experiments a) to evaluate the performances of the selected machine learning models executed in an ensemble manner using multi-threading, b) to evaluate PhishHaven against AI-generated and Simple Phishing URLs, and c) to compare PhishHaven against some of the existing lexical features-based simple phishing URLs detection systems.

Our approach of parallel training and testing of the ensemble machine learning models enables PhishHaven to speed-up the process of URL classification. To evaluate the performances of our selected machine learning models which are appropriate in our case, we evaluate them against Precision, Recall and F1-measure as shown in Table 6.

It can be seen in Table 6 that among the selected machine learning models, Support Vector Machines performs significantly well having 97.68% Precision, 97.63% Recall and

TABLE 6. Performances of selected machine learning models.

Machine Learning Models	Precision	Recall	F1-measure
AdaBoost Classifier	94.35	93.93	93.95
Bagging Classifier	96.84	96.77	96.78
Decision Trees	96.79	96.75	96.76
Extra Tree Classifier	95.22	95.10	95.11
Gradient Boosting Classifier	94.40	93.73	93.75
K-Nearest Neighbour	94.01	93.77	93.73
Logistic Regression	96.76	96.71	96.72
Neural Networks	96.66	96.64	96.65
Random Forest	96.79	96.75	96.89
Support Vector Machines	97.68	97.63	97.64

TABLE 7. PhishHaven evaluation for AI-generated Phishing URLs.

Performance based	Precision	98.08%
	Recall	98.06%
	Accuracy	98.08%
	F1-measure	98.06%
Efficiency based	TPR	96.74%
	FPR	0.82%
	TNR	99.17%

TABLE 8. PhishHaven evaluation for simple phishing URLs.

Performance based	Precision	98.00%
	Recall	97.95%
	Accuracy	98.00%
	F1-measure	97.96%
Efficiency based	TPR	97.00%
	FPR	0.83%
	TNR	99.17%

97.64% F1-measure. The reason Support Vector Machines work significantly well in our case is that they perform binary classification efficiently by drawing decision boundary in hyperplane between two classes. Table 10 describes the parameters setup of our selected ten machine learning models in our experimental setup.

However, we proposed and employed ensemble-based machine learning through multi-threading for final classification. Therefore, Figure 10 illustrates how the performances of our selected machine learning models contribute altogether to the overall performance and efficiency of our detection system.

We test our detection system for two different types of URLs' detection, i.e., for AI-generated Phishing URLs and for Simple Phishing URLs. Our evaluation measures, as shown in Table 7 and Table 8, signify that our detection system is capable enough to detect efficiently not only AI-generated Phishing URLs but also Simple Phishing URLs and Normal URLs.

To evaluate the performance of our detection system, we choose the following measures: Precision, Recall, Accuracy, and F-1measure. To analyze the efficiency of our detection system, we employ Sensitivity (TPR), Fall-out (FPR) and Specificity (TNR).

TABLE 9. PhishHaven comparison with some of the existing lexical-based detection systems for simple phishing URLs.

Models	Precision	Accuracy
Marchal, Samuel, et al. [21]	98.44%	94.91%
Rao, R. S., Vaishnavi, T., & Pais, A. R. [23]	94.22%	90.28%
Ubing, Alyssa Anne, et al. [24]	93.50%	95.40%
Sahingoz, Ozgur Koray, et al. [25]	97.00%	97.98%
Gajera, Kishan, et al. [27]	97.56%	98.77%
PhishHaven *[our proposed solution]*	98.00%	98.00%

To further evaluate the performance of PhishHaven in terms of Simple Phishing URLs detection, we compare PhishHaven with some of the existing lexical-based simple phishing URLs detection systems. In Table 9, we can see that PhishHaven outperforms the existing state-of-the-art detection systems designed for detecting simple phishing URLs. From Table 9, we can also see that in [27] they have slightly higher accuracy than PhishHaven because their dataset is smaller than ours, i.e., we took a dataset with a wider range of cases including a variety of significant cases.

VI. RELATED WORK

Phishing detection systems usually utilize several techniques for detecting and predicting phishing sites. The main techniques are as follows:

A. LISTS AND HEURISTICS-BASED TECHNIQUES

In these techniques, detection mechanisms employ whitelists or blacklists and a set of rules to compare and classify a URL either as phishing or normal. A major drawback of these approaches is that they completely fail in detecting newly generated phishing sites called zero-day phishing sites. Furthermore, they require continuous update of lists and rules. Also, websites which are similar in terms of URL contents to those in blacklists or whitelists, and websites which are similar in terms of appearance to those set heuristics can be easily misclassified.

In [15], the authors presented a phishing URLs detection approach based on string-matching which is heavily dependent on blacklists, which consumed significant computation cost.

On the other hand, in [16], the authors proposed a solution to improve the blacklisting-based phishing URLs detection systems. Their approach was not effective against zero-day attacks. In addition to this, their approach used a significant amount of computation time with relatively less outcome.

B. CONTENT-BASED TECHNIQUES

This approach analyzes the content of the webpage to classify the respective page either as phishing or normal. A main limitation which makes this approach not only computationally inefficient but also not a viable technique for most of the scenarios is that it requires either source code or the entire content of the website, i.e., images or text for performing features extraction and analysis process.

TABLE 10. Parameters setup of selected machine learning models.

Approaches	Machine Learning Models	Parameters
Boosting-based	AdaBoost Classifier	*base_stimator=None, n_stimators=50, learning_ate=1.0, algorithm='SAMME.R', random_tate=None*
	Gradient Boosting Classifier	*loss='deviance',learning_ate=0.1,n_stimators=100, subsample=1.0,criterion='friedman_se',min_amples_plit=2, min_amples_eaf=1,min_eight_raction_eaf=0.0,max_epth=3, min_mpurity_ecrease=0.0,min_mpurity_plit=None,init=None, random_tate=None,max_eatures=None,verbose=0,max_eaf_odes=None, warm_tart=False,presort='auto',validation_raction=0.1,n_ter_o_hange=None,tol=0.0001*
Non-Learning-based	Bagging Classifier	*base_stimator=None,n_stimators=10,max_amples=1.0, max_eatures=1.0,bootstrap=True,bootstrap_eatures=False, oob_core=False,warm_tart=False,n_obs=None,random_tate=None, verbose=0*
	Decision Trees	*criterion='gini',splitter='best',max_epth=None,min_amples_plit=2, min_amples_eaf=1,min_eight_raction_eaf=0.0,max_eatures=None, random_tate=None,max_eaf_odes=None,min_mpurity_ecrease=0.0, min_mpurity_plit=None,class_eight=None,presort=False*
	Extra Tree Classifier	*n_stimators=100,criterion='gini',max_epth=None,min_amples_plit=2, min_amples_eaf=1,min_eight_raction_eaf=0.0,max_eatures='auto', max_eaf_odes=None,min_mpurity_ecrease=0.0,min_mpurity_plit=None, bootstrap=False,oob_core=False,n_obs=None,random_tate=None, verbose=0,warm_tart=False,class_eight=None,ccp_lpha=0.0,max_amples=None*
	K-Nearest Neighbour	*n_eighbors=5,weights='uniform',algorithm='auto',leaf_ize=30, p=2,metric='minkowski',metric_arams=None,n_obs=None*
	Random Forest	*n_stimators=100,criterion='gini',max_epth=None,min_amples_plit=2, min_amples_eaf=1,min_eight_raction_eaf=0.0,max_eatures=None, max_eaf_odes=None,min_mpurity_ecrease=0.0,min_mpurity_plit=None,bootstrap=True, oob_core=False,n_obs=None,random_tate=None,verbose=0,warm_tart=False, class_eight=None*
Learning-based	Logistic Regression	*penalty='l2',dual=False,tol=0.0001,C=1.0,fit_ntercept=True,intercept_caling=1, class_eight=None,random_tate=None,solver='saga',max_ter=100,multi_lass='ovr', verbose=0,warm_tart=False,n_obs=None,l1_atio=None*
	Neural Networks	*solver='lbfgs',alpha=1e-5,hidden_ayer_izes=(10, 10),random_tate=1*
	Support Vector Machines	*gamma = 'scale*

In [17], the authors applied similarity in CSS(Cascading Style Sheets) features, i.e., visual features. Their proposed scheme consumed more time as a whole. In [18], the authors proposed a phishing webpage detection mechanism. Their mechanism noticeably outperformed, but consumed a lot of time. Furthermore, their approach of using stacked models involves GBDT(Gradient Boosting Decision Tree) for making initial and final predictions, added a biased factor in the final predictions.

The approach proposed in [19] utilized hyperlinks features extracted from pages' source codes. Since their approach completely relied on the analysis of features extracted from the source code, therefore, the longer the source code, the higher the time-complexity.

C. THIRD-PARTY-BASED TECHNIQUES

Some detection mechanisms use third-party-based features and services. The main drawback of this approach is high error rate in terms of misclassification. The main reason for misclassification is that they heavily rely on either the age of a domain or the number of occurrences in search results. There are likely chances in this approach that newly setup legitimate sites can be misclassified as phishing sites. In addition to this, there are chances that the third-party can be biased or hijacked, e.g., DNS(Domain Name System) spoofing, etc.

In [20], the authors conducted a study on phishing URLs detection using both lexical and external (third-party-based) features. Their proposed solution named as ''PhishDef'' demonstrated that it performed significantly accurate using only lexical features.

D. LEXICAL FEATURES-BASED TECHNIQUES

This URL classification technique turns out to be a more promising approach. This technique totally relies on URLs for features extraction. These features can be count-based features, binary features, blacklisted words, etc. It is also computationally efficient as it only takes a URL into consideration, i.e., a line comprises of alphanumeric characters and symbols.

In [21], the authors designed a phishing detection system which yielded an accuracy of 94.91% employing lexical feature analysis. In [22], the authors enhanced the performance of phishing URLs detection system through lexical features. The proposed model in [23] achieved a noticeable accuracy with the lexical features and consumed relatively less time since it was independent of any third-party services and source code analysis. In [24], the authors conducted a

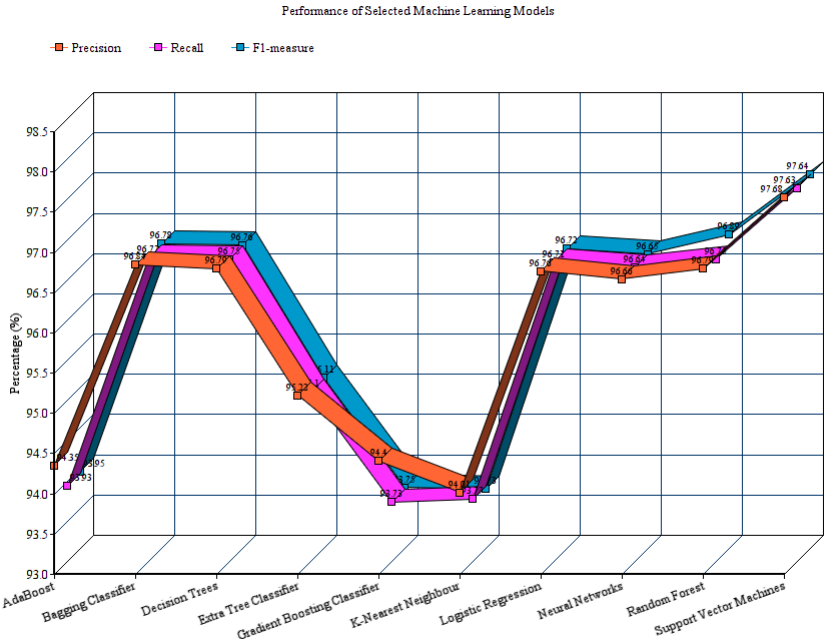

FIGURE 10. Graphical illustration of selected machine learning models' performance.

TABLE 11. Key limitations of features extraction techniques.

Techniques	Lists and Heuristics-based	Content-based	Third-party-based	Lexical-based
Limitations	• Failed to detect zero-day attacks • Requires a continuous update of lists and rules • Misclassifies URLs based on appearance or rules with blacklists and whitelists or set rules respectively • Requires 'n' time complexity for performing a comparison or matching	• Extracts content of webpages, hence computationally inefficient • Non-proactive approach • Requires source codes or the entire page content of the website • Requires 'n' time complexity for performing a comparison or matching	• Third-party itself can be hijacked computationally expensive • Higher chances of misclassifications	• Requires 'n' time complexity for lexical features extraction

study on improving the accuracy of the phishing detection systems through features selection and ensemble learning methodology. Their experimental results showed an accuracy of 95%. The detection system implemented in [25] used seven different machine learning algorithms. Their approach and types of selected features overcome many issues like language and third-party dependency, as well as real-time and zero-day attacks detection.

E. HYBRID FEATURES-BASED TECHNIQUES

In [26], the authors applied several techniques together such as whitelist, external features (third-party services), page contents, and TF-IDF techniques. Though the authors proposed the solution to improve the maintenance process of a blacklist, the classification techniques of HTML page content, external features and TF-IDF inherited their limitations to their proposed solution as well. Also, the solution proposed in [27] employed a combination of various techniques. Their system heavily relied on source code analysis, verification from whitelists, third-party-based services and page similarity using screenshots for detection purpose. Hence, this made their solution more time-consuming and less efficient. On the other hand, in [28], the authors employed lexical as well as host-based features to detect phishing URLs. Though the authors proposed a solution to complement blacklisting and heuristic-based detection systems but their proposed solution used host-based properties, hence inherited various limitations of host-based techniques.

From the above discussed studies and comparison shown in Table 11, we concluded that lexical features-based techniques are more efficient with fewer limitations. Therefore, we design our detection system based on lexical features extraction and analysis.

VII. CONCLUSION

In this paper, we proposed PhishHaven, the first AI-generated Phishing URLs detection system based on ensemble machine learning. Our proposed system is based on lexical features analysis. We also introduced URL HTML Encoding as a lex-

ical feature to boost our detection system in proactive and on-the-fly detection of URLs. We further introduced a URL Hit approach for detecting and classifying tiny URLs. In addition to this, we presented a new paradigm for ensemble-based machine learning models execution. Our proposed new paradigm executes ensemble-based machine learning models in parallel using multi-threading technique, and results in real-time detection by significant speed-up in the classification process. For final classification, we employed an unbiased voting method.

We evaluated our solution both theoretically and experimentally. In theoretical analysis, we proved that our solution can detect tiny URLs as well as future AI-generated Phishing URLs based on our selected lexical features with 100% accuracy. Experimental analyses were conducted for two cases, i.e., AI-generated and Simple Phishing URLs.The dataset in the first case consists of AI-generated Phishing URLs and Normal URLs. The dataset in the second case consists of Simple Phishing URLs and Normal URLs. For the first case, the results showed a significantly high accuracy and F1-measure of 98%, while securing 97% TPR and 99.17% TNR with noticeably low fall-out of 0.8% FPR. For the second case, the results also showed a significantly high accuracy and precision of 98%, outperforming the other existing simple phishing URLs detection systems. Therefore, we can conclude that the proposed solution efficiently addresses the detection of AI-generated Phishing URLs in the forthcoming future as well as Simple Phishing URLs prevalent these days.

Although PhishHaven has achieved a significant accuracy in classifying both AI-generated and human-crafted phishing URLs, it has a limitation that it can detect only those AI-generated Phishing URLs which consist of lexical features and patterns similar to that of DeepPhish [1]. It is because, to the best of our knowledge, DeepPhish [1] is the only AI-based system designed to generate phishing URLs.

We have some future work and directions as follows:

1) PhishHaven can be further enhanced by incorporating unsupervised learning, i.e., deep learning models
2) The efficiency of PhishHaven can be further improved in the following ways:
    a) Based on our chosen metric, we can filter outperforming models and only use those models for making predictions. Hence, with this kind of model reduction, we can reduce the computation cost in terms of multi-threading.
    b) In continuation of the previous point, we can also extract the weights assigned by outperforming models to extract features and consider only those features. Hence with this type of approach for features reduction, we can reduce the computation process and cost in the Features Extractor subcomponent.
3) By applying multi-threading technique at an input unit (i.e., at the very initial point where PhishHaven takes a URL as an input), we can work on multiple URLs simultaneously, hence incorporating the scalibility factor in PhishHaven for classifying multiple URLs at a time.

REFERENCES

[1] A.-C. Bahnsen, I. Torroledo, D. Camacho, and S. Villegas, "DeepPhish: Simulating malicious AI," *APWG Symp. Electron. Crime Res. (eCrime)*, 2018, pp. 1–8.
[2] M. Zolanvari, M. A. Teixeira, L. Gupta, K. M. Khan, and R. Jain, "Machine learning-based network vulnerability analysis of industrial Internet of Things," *IEEE Internet Things J.*, vol. 6, no. 4, pp. 6822–6834, Aug. 2019.
[3] C. Peng, H. Sun, M. Yang, and Y.-L. Wang, "A survey on security communication and control for smart grids under malicious cyber attacks," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 49, no. 8, pp. 1554–1569, Aug. 2019.
[4] M. Cui, J. Wang, and M. Yue, "Machine learning-based anomaly detection for load forecasting under cyberattacks," *IEEE Trans. Smart Grid*, vol. 10, no. 5, pp. 5724–5734, Sep. 2019.
[5] M. Al Janaideh, E. Hammad, A. Farraj, and D. Kundur, "Mitigating attacks with nonlinear dynamics on actuators in cyber-physical mechatronic systems," *IEEE Trans. Ind. Informat.*, vol. 15, no. 9, pp. 4845–4856, Sep. 2019.
[6] S. Soltan, Mihalis-Yannakakis, and Gil-Zussman, "REACT to Cyber-Physical Attacks on Power grids," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 46, no. 2, pp. 50–51, 2019.
[7] Anti-Phishing Working Group (APWG). (2019). *Phishing Activity Trends Report—Third Quarter 2019*. https://docs.apwg.org/reports/apwg_trends_report_q3_2019.pdf
[8] T. Thomas, P. A. Vijayaraghavan, and S. Emmanuel, "Machine Learning and Cybersecurity," in *Machine Learning Approaches in Cyber Security Analytics*. Singapore: Springer, 2020.
[9] M. T. Suleman and S. M. Awan, "Optimization of URL-based phishing websites detection through genetic algorithms," *Autom. Control Comput. Sci.*, vol. 53, no. 4, pp. 333–341, Jul. 2019.
[10] E. Zhu, Y. Chen, C. Ye, X. Li, and F. Liu, "OFS-NN: An effective phishing websites detection model based on optimal feature selection and neural network," *IEEE Access*, vol. 7, pp. 73271–73284, 2019.
[11] D. Goel and A. K. Jain, "Mobile phishing attacks and defence mechanisms: State of art and open research challenges," *Comput. Secur.*, vol. 73, pp. 519–544, Mar. 2018.
[12] DARKTRACE. (2018). *The Next Paradigm Shift AI-Driven Cyber-Attacks*. [Online]. Available: https://www.darktrace.com/en/resources/wp-ai-driven-cyber-attacks.pdf
[13] M. Brundage, "Limitations and risks of machine ethics," *J. Exp. Theor. Artif. Intell.*, vol. 26, no. 3, pp. 355–372, 2014.
[14] C. C. Galland, "The limitations of deterministic Boltzmann machine learning," *Netw., Comput. Neural Syst.*, vol. 4, no. 3, pp. 355–379, Jan. 1993.
[15] D. Abraham and N. S. Raj, "Approximate string matching algorithm for phishing detection," in *Proc. Int. Conf. Adv. Comput., Commun. Informat. (ICACCI)*, Sep. 2014, pp. 2285–2290.
[16] P. Prakash, M. Kumar, R. R. Kompella, and M. Gupta, "PhishNet: Predictive blacklisting to detect phishing attacks," in *Proc. IEEE INFOCOM*, Mar. 2010, pp. 1–5.
[17] J. Mao, P. Li, K. Li, T. Wei, and Z. Liang, "BaitAlarm: Detecting phishing sites using similarity in fundamental visual features," in *Proc. 5th Int. Conf. Intell. Netw. Collaborative Syst.*, Sep. 2013, pp. 790–795.
[18] Y. Li, Z. Yang, X. Chen, H. Yuan, and W. Liu, "A stacking model using URL and HTML features for phishing webpage detection," *Future Gener. Comput. Syst.*, vol. 94, pp. 27–39, May 2019.
[19] A. K. Jain and B. B. Gupta, "A machine learning based approach for phishing detection using hyperlinks information," *J. Ambient Intell. Hum. Comput.*, vol. 10, no. 5, pp. 2015–2028, May 2019.
[20] A. Le, A. Markopoulou, and M. Faloutsos, "PhishDef: URL names say it all," in *Proc. IEEE INFOCOM*, Apr. 2011, pp. 191–195.
[21] S. Marchal, J. Francois, R. State, and T. Engel, "PhishStorm: Detecting phishing with streaming analytics," *IEEE Trans. Netw. Service Manage.*, vol. 11, no. 4, pp. 458–471, Dec. 2014.
[22] H. Tupsamudre, A. K. Singh, and S. Lodha, "Everything is in the name—A URL based aApproach for phishing detection," in *Proc. Int. Symp. Cyber Secur. Cryptography Mach. Learn.* Cham, Switzerland: Springer, 2019, pp. 231–248.
[23] R. S. Rao, T. Vaishnavi, and A. R. Pais, "CatchPhish: Detection of phishing Websites by inspecting URLs," *J. Ambient Intell. Hum. Comput.*, vol. 11, no. 2, pp. 813–825, Feb. 2020.

[24] A. A. Ubing, S. Kamilia, A. Abdullah, N. Jhanjhi, and M. Supramaniam, "Phishing Website detection: An improved accuracy through feature selection and ensemble learning," *Int. J. Adv. Comput. Sci. Appl.*, vol. 10, no. 1, pp. 252–257, 2019.
[25] O. K. Sahingoz, E. Buber, O. Demir, and B. Diri, "Machine learning based phishing detection from URLs," *Expert Syst. Appl.*, vol. 117, pp. 345–357, Mar. 2019.
[26] C. Whittaker, B. Ryner, and M. Nazif, "Large-scale automatic classification of phishing pages," in *Proc. NDSS*, 2010. [Online]. Available: http://www.isoc.org/isoc/conferences/ndss/10/pdf/08.pdf
[27] K. Gajera, M. Jangid, P. Mehta, and J. Mittal, "A novel approach to detect phishing attack using artificial neural networks combined with pharming detection," in *Proc. 3rd Int. Conf. Electron., Commun. Aerosp. Technol. (ICECA)*, Jun. 2019, pp. 196–200.
[28] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond blacklists: Learning to detect malicious Web sites from suspicious URLs," in *Proc. 15th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2009, pp. 1245–1254.
[29] R. Fielding and J. Reschke, *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*, document RFC 7231, Jun. 2014. [Online]. Available: https://www.rfc-editor.org/info/rfc7231
[30] N. Delmotte. (Jun. 2008). *HTML URL-Encoding Reference*. [Online]. Available: https://www.eso.org/~ndelmott/url_encode.html
[31] Schapire, Robert E., "Explaining Adaboost," *Empirical Inference*. Berlin, Germany: Springer, 2013, pp. 37–52.
[32] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm," in *Proc. Int. Conf. Mach. Learn.*, vol. 96. 1996, pp. 148–156.
[33] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Ann. Statist.*, vol. 29, no. 5, pp. 1189–1232, Oct. 2001.
[34] Perlich, Claudia, Foster Provost, and Jeffrey S. Simonoff, "Tree induction vs. Logistic regression: A learning-curve analysis," *J. Mach. Learn. Res.*, vol. 4, pp. 211–255, Jun. 2003.
[35] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.
[36] Geurts, Pierre, Damien Ernst, and Louis Wehenkel, "Extremely randomized trees," *Mach. Learn.*, vol. 63, no. 1, pp. 3–42, 2006.
[37] G. Guo, H. Wang, D. Bell, Y. Bi, and K. Greer, "KNN model-based approach in classification," *On Move to Meaningful Internet Systems*. Berlin, Germany: Springer, 2003, pp. 986–996.
[38] S. Dreiseitl and L. Ohno-Machado, "Logistic regression and artificial neural network classification models: A methodology review," *J. Biomed. Informat.*, vol. 35, nos. 5–6, pp. 352–359, 2002.
[39] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, 1995.
[40] M. W. Gardner and S. R. Dorling, "Artificial neural networks (the multilayer perceptron)," *Atmos. Environ. Rev. Appl.*, vol. 32, nos. 14–15, pp. 2627–2636, 1998.
[41] *DNS Twist*. Accessed: Sep. 4, 2019. [Online]. Available: https://github.com/elceef/dnstwist
[42] *Alexa*. Accessed: Sep. 4, 2019. [Online]. Available: https://www.alexa.com
[43] *PhishTank*. Accessed: Sep. 4, 2019. [Online]. Available: https://www.phishtank.com

MARIA SAMEEN received the B.S. degree in computer science from the Institute of Business Administration (IBA), Karachi, Pakistan, in 2018. She is currently a Researcher with the Information Security and Machine Learning Laboratory, Gachon University, South Korea. Her research interests include cybersecurity, network and system security, malware analysis, machine learning, artificial intelligence, and the Internet-of-Things security.

KYUNGHYUN HAN received the B.S. degree in computer engineering and the M.S. degree in computer engineering from Hongik University, South Korea, in 2015 and 2017, respectively. He is currently a Researcher with the Information Security and Machine Learning Laboratory, Hongik University. His research interests include cyber security, machine learning, and blockchain.

SEONG OUN HWANG (Senior Member, IEEE) received the B.S. degree in mathematics from Seoul National University, in 1993, the M.S. degree in information and communications engineering from the Pohang University of Science and Technology, in 1998, and the Ph.D. degree in computer science from the Korea Advanced Institute of Science and Technology, South Korea. He worked as a Software Engineer with LG-CNS Systems, Inc., from 1994 to 1996. He also worked as a Senior Researcher with the Electronics and Telecommunications Research Institute (ETRI), from 1998 to 2007. He worked as a Professor with the Department of Software and Communications Engineering, Hongik University, from 2008 to 2019. He is currently a Professor with the Department of Computer Engineering, Gachon University. He is also an Editor of *ETRI Journal*. His research interests include cryptography, cybersecurity, and artificial intelligence.