ABSTRACT

Title of Dissertation: A COMPREHENSIVE EVALUATION OF FEATURE-BASED MALICIOUS WEBSITE DETECTION

John F. McGahagan IV, Doctor of Philosophy, 2020

Dissertation directed by: Professor Michel Cukier
Department of Mechanical Engineering

Although the internet enables many important functions of modern life, it is also a ground for nefarious activity by malicious actors and cybercriminals. For example, malicious websites facilitate phishing attacks, malware infections, data theft, and disruption. A major component of cybersecurity is to detect and mitigate attacks enabled by malicious websites. Although prior researchers have presented promising results – specifically in the use of website features to detect malicious websites – malicious website detection continues to pose major challenges. This dissertation presents an investigation into feature-based malicious website detection. We conducted six studies on malicious website detection, with a focus on discovering new features for malicious website detection, challenging assumptions of features from prior research, comparing the importance of the features for malicious website detection, building and evaluating

detection models over various scenarios, and evaluating malicious website detection models across different datasets and over time. We evaluated this approach on various datasets, including: a dataset composed of several threats from industry; a dataset derived from the Alexa top one million domains and supplemented with open source threat intelligence information; and a dataset consisting of websites gathered repeatedly over time. Results led us to postulate that new, unstudied, features could be incorporated to improve malicious website detection models, since, in many cases, models built with new features outperformed models built from features used in prior research and did so with fewer features. We also found that features discovered using feature selection could be applied to other datasets with minor adjustments. In addition: we demonstrated that the performance of detection models decreased over time; we measured the change of websites in relation to our detection model; and we demonstrated the benefit of re-training in various scenarios.

A COMPREHENSIVE EVALUATION OF FEATURE-BASED MALICIOUS WEBSITE DETECTION

by

John F. McGahagan IV

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2020

Advisory Committee:

Professor Michel Cukier, Chair
Professor Jennifer Golbeck, Dean's Representative
Assistant Professor Katrina Groth
Professor Jeffrey Herrmann
Professor Mohammad Modarres

Acknowledgements

I would like to thank my advisor, Dr. Michel Cukier, for his guidance, mentorship, and feedback in this pursuit. Thank you to Darshan Bhansali for his collaboration and for sharing his data science expertise with me. I would also like to express my gratitude to Dr. Margaret Gratian and Ciro Pinto-Coelho for their constructive feedback on the published works derived from this dissertation. Thank you to Robert Jenquin for his outside perspective on this work. An additional thanks to Karl and Vicki Gumtow for their support of this academic pursuit. Thank you to Tim McGahagan and the countless family members and friends for your support in this journey. Last, but not least, I would like to thank my parents Diane and John McGahagan III for their continued love, encouragement, and words of wisdom in this endeavor and in life.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AB	Adaptive Boosting
ACC	Accuracy
ADASYN	Adaptive Synthetic Sampling
AGD	Algorithmically Generated Domain
ASCII	American Standard Code for Information Interchange
AST	Abstract Syntax Tree
AUC	Area Under the Curve
BC	Bagging Classifier
C2	Command and Control
CART	Classification and Regression Trees
CPT	Control Protocol Template
DDoS	Distributed Denial of Service
DGA	Domain Generation Algorithm
DNS	Domain Name System
DOM	Document Object Model
DoS	Denial of Service
ET	Extra Trees
FNR	False Negative Rate
FPR	False Positive Rate
GB	Gradient Boosting
GLM	Generalized Linear Model
HGD	Human-Generated Domain
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
JSAND	JavaScript Anomaly-based Analysis and Detection
KNN	K-Nearest Neighbor
KS	Kolmogorov-Smirnov
LR	Logistic Regression
MDN	Mozilla Developer Network
MIME	Multipurpose Internet Mail Extensions
ML-kNN	Multi-Label K-Nearest Neighbor
NN	Neural Networks
OoD	Out of Domain
PCA	Principal Component Analysis
Prec	Precision
Rec –	Recall
RF –	Random Forest
RFE	Recursive Feature Elimination
ROC	Receiver Operating Characteristic
RRP	Request Response Pair
SMOTE	Synthetic Minority Over-Sampling Technique
SOC –	Security Operations Center
SSL	Secure Sockets Layer
SVM	Support Vector Machine

TCP	Transmission Control Protocol
TF- IDF	Term Frequency – Inverse Document Frequency
TLD –	Top-Level Domain
TNR –	True Negative Rate
TPR –	True Positive Rate
t-SNE	T-distributed Stochastics Neighbor Embedding
V –	Voting
VIF –	Variance Inflation Factor
XGB –	XgBoost

Chapter 1: Introduction

1.1 Background and Motivation

1.1.1 The Impact of Cybersecurity

The internet has changed the way we live and work. Over the years, more and more aspects of human life have become reliant on the internet. From organizing our personal lives to banking and entertainment, the internet plays a large part in how we, as humans, exchange information. Pew Research [1] reported that as of 2019, 90% of all U.S. adults used the internet. As of June 2019, roughly 58.8% of the world's population (4.536 billion people) use the internet, up from 5.8% in December 2000 [2]. In addition to the increasing numbers of people accessing the internet, the internet has a large financial impact and is a common place to conduct business. Digital Commerce 360's [3] analysis of U.S. Department of Commerce data estimated that consumers spent $513.61 billion dollars online in 2018, up 14.2% from online spending in 2017. Although the internet has added efficiency to our lives by facilitating communication and changing the way we live, the emergence of the internet has also created an opportunity for criminals and other nefarious actors to conduct malicious activity.

The threat from malicious cyber actors is so great that corporate and government entities allocate substantial budgets toward detecting, preventing, and remediating cyber threats. The financial impact on corporations is large, with Cavusoglu et al. [4] reporting in their study of the financial impact on firms with breaches that firms in their sample lost 2.1% (or $1.65 billion) of their market capitalization within two days of announcement of a cyber breach. Experts [5] have projected that more than $1 trillion dollars will be spent on digital security globally on an annual basis. Large corporations such as Bank of

America and J.P. Morgan Chase spend as much as $500 million each year on cybersecurity [6]. Breaches and incidents can be large in terms of the number of people and accounts affected and in terms of the loss of money due to litigation and business impact. The Yahoo! breach in 2014 resulted in theft of personal information from more than 500 million accounts [7]. The Epsilon hack had a financial impact totaling upwards of $4 billion [8]. Dyn, which was the victim of a DDoS attack by the Mirai botnet in 2016 [9], lost roughly 8% of its customers due to the impact of the attack [10]. Given the potential for tremendous repercussions from cyber threats, industry and government entities recognize the need to protect their assets and their businesses against such attacks.

Cyber-attack goals depend on the motivations of the attacker. Attackers commonly seek to either steal information, infect a victim's network, or disrupt a victims' ability to function. Stealing personal information may enable an attacker to misuse the victim's identity, resulting in financial loss. Information theft also may facilitate blackmailing of the victim. Disruption can harm the victim's reputation or simply stop victims from performing their functions. Infection can facilitate information stealing and disruption.

1.1.2 Websites as Attack Enablers

Malicious actors can conduct many types of attacks. The most prevalent attacks include phishing [11], drive-by downloads [12], denial of service (DoS) [13], or other kinds of attacks caused by infection. Phishing occurs when an attacker tries to "trick" a victim into entering personal or sensitive information, visiting a malicious website, or opening or interacting with a malicious email or link. Phishing detection is typically

focused on examining a website or email for suspicious indicators. Drive-by downloads occur when a user visits a website and falls victim to malicious code execution that typically occurs when the website is being rendered. JavaScript [14] on the website is a common attack vector for drive-by downloads. DoS attacks can occur from compromised devices or from specific malicious domains. Infection can take many forms, with the most sophisticated form resulting in command and control (C2) [15] with a malicious website or server. C2 activity occurs when an attacker has compromised a network or asset in the network and is running malware on the compromised network. This malware typically receives commands or exfiltrates data from or to the C2 infrastructure. The C2 infrastructure can specify actions to take inside the victim's network. To communicate with this malware, the attacker needs a malicious website or domain. For each of the attacks we have discussed thus far, attackers also require a website or domain from which to conduct the malicious activity. Detecting malicious websites and blocking communication with them is a major component of cybersecurity.

1.1.3 The Case for New Detection Techniques

Cyber threats and cyberattacks increase in complexity over time, making it a challenge to detect them. There is a constant battle between attackers and defenders, both of which are looking for an advantage. Defenders are at an inherent disadvantage, given the need to consider all aspects of their systems and defend each one properly. An attacker, on the other hand, has only to identify a weakness or two in order to conduct an attack. Furthermore, defenders must account for unknown vulnerabilities that may exist in their systems. These vulnerabilities are often in third-party software that defenders did not create. Unfortunately, vulnerabilities are common and sometimes disclosed without

remediation mechanisms or patches. For example, Risk Based Security [16] reported that 22,000 vulnerabilities were disclosed in 2018 without fixes being provided, a trend that is expected to continue.

Defenders have access to a number of tools for detecting and preventing attacks, including anti-virus software, network intrusion detection systems, denylists, threat intelligence, etc. Over the years, these tools have evolved to keep up with threats. For example, the Morris Worm, an early self-propagating virus, took advantage of a security flaw in the *sendmail* function in Unix [17]. Such security threats encouraged the creation of anti-virus software. Early anti-virus software detected viruses by examining hashes of files or strings specific to known malware. However, once anti-virus tools began detecting viruses with hashes and strings, malware began to adapt by creating variants with different binary signatures. At this point, detecting malware with hashes alone became infeasible. The anti-virus community adjusted, beginning to detect malware families instead of specific files and binaries by using signatures that applied to several binaries instead of to a single binary. Evolution by attackers and defenders is natural and will continue. With this research, we aimed to assist the detection community by exploring additional mechanisms and insights for detecting malicious websites.

1.1.4 The Current State of Malicious Website Detection

The techniques for detecting malicious websites have evolved over the years. A common method that is still used today for validation involves visiting a website to analyze the web response, analyzing the instructions executed when rendering the webpage, and comparing the observations to known malicious behavior. Researchers also can instrument their systems to look for other potential malicious activity that results

from visiting a webpage. For example, if a user fetches a webpage in Firefox and observes an unexpected event such as an attempted registry change (on Windows) [18] or observes an unlikely file change, this may be an indication of a malicious webpage. Although this method can be used for validation, it faces two challenges. First, it is time consuming and requires additional resources for visiting each website, recording effects, and verifying whether the website is malicious. Websites can change very quickly, making this effort more complicated. Secondly, this approach may miss malicious websites with malicious behavior that does not match a known signature [19].

Another common technique for detecting malicious websites is to collect "features" and use them to create signatures or models for malicious website detection. This approach is the foundation of the research in this dissertation. In this paradigm, features or observational characteristics – the Uniform Resource Locator (URL) [20] structure or Hypertext Markup Language (HTML) [21] tags on the page, for example – are extracted from a website. These features are then turned into rules, signatures, or models to detect other malicious websites. While this approach provides less certainty regarding a website's maliciousness (the results are typically presented as a probability of the website being malicious), the approach can capture commonalities that may exist among malicious websites, thereby facilitating detection. Although using features to detect malicious websites is increasingly common, it does have challenges. To perform this approach, the researcher or practitioner must choose which features to collect. Prior researchers have typically collected well-known features to detect malicious websites, but rarely re-evaluated whether those features were still useful or whether other features could also be used to identify malicious websites. Additionally, the most successful

studies tended to evaluate an approach on a dataset consisting of a single threat. Narrowing experiments to a single type of threat served to focus the research on that specific attack, but also required *a priori* knowledge of the threat, making it less applicable when *a priori* knowledge is unavailable.

1.2 Research Scope

1.2.1 Detecting Malicious Websites

Although attacks can be detected in many ways, we focused this study on detecting a fundamental enabler of malicious activities – the website. The primary ways that websites can be misused include: 1) tricking a user into entering sensitive information or "faking" a legitimate website (creating what is also known as a phishing website); 2) delivering malicious content; and 3) serving as a communication point to malware and other malicious software. These misuses are illustrated in Fig. 1-1, 1-2, and 1-3 below.

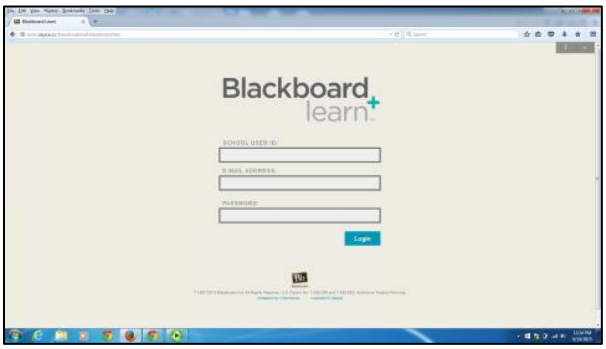


Fig. 1-1. An example of a phishing website from Lehigh.edu (Image courtesy of Pixabay [22].)

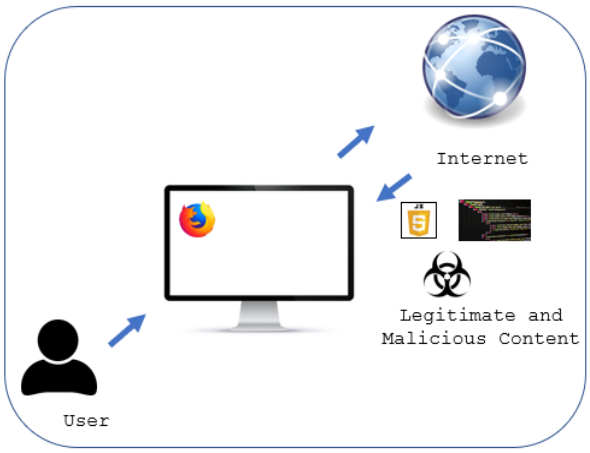


Fig. 1-2. A simplified view of drive-by down-load infections (Images courtesy of Pixabay [22].)

Fig. 1-3. A simplified view of C2 (Images courtesy of Pixabay [22].)

1.2.2 Identifying and Comparing New Features for Malicious Website Detection

Although using features that have already demonstrated potential for detecting malicious websites is a popular approach in prior research, there has been little emphasis on finding new features for malicious website detection. For example, the `<iframe>` has been a feature used for malicious website detection since at least 2008 [23]. Similarly, the number of "." characters in a URL is a feature that has been used for malicious website

detection since at least 2007 [24]. In this research, we evaluated and identified new features for malicious website detection.

In addition to identifying new features for malicious website detection, we also quantified and compared the performance of the new features for detecting malicious websites and compared it to those of features from prior research. Specifically, we compared the rank and importance of those features (both new features and those identified in prior research). We determined the importance of each feature by defining by how much it contributed to and influenced the performance of the malicious website detection models it produced. Additionally, we gathered performance metrics on detection models built with learning algorithms [25] and with features identified in our approach, as well as with models built from features from prior research.

1.2.3 Evaluating our Approach over Multiple Scenarios

We then went on to evaluate the identified features and their respective detection models across a variety of scenarios. Scenarios included sampling to balance our dataset, feature transformation, and principal component analysis (PCA) [28] to create meta-features and components. Such evaluations increased assurance that our approach, results, and observations were not specific to a single experimental scenario and would prove valid should future researchers replicate our study with different setups.

We explored sampling scenarios to account for our dataset imbalance and explored feature transformation to evaluate whether combinations of features could improve malicious website detection. We also experimented with the `class weight` parameter as another method of balancing our datasets. Lastly, we performed hyperparameter tuning and cross-validation of our models [26] in order to achieve the

best performance metrics for our detection models. Parameters were specific to the respective classifiers.

1.2.4 Bridging a Gap Between Research and Industry

With this research, we endeavored to bridge research and industry gaps in malicious website detection. Although existing research has demonstrated success in various studies, the problem of malicious website detection persists. There are differences, of course, between an environment in which research is conducted and an operational scenario. First, research operates on in-depth knowledge of the malicious dataset under study, a factor that often influences the features collected for detection. For example, researchers who focused on detecting phishing websites would collect HTML and other visual features from webpages since these have been demonstrated to detect phishing attacks. In an operational scenario, however, the goal is to prevent the network from accessing malicious websites regardless of their nature. To more closely replicate an operational environment, we used datasets consisting of common threats, specifically: phishing, drive-by downloads, and C2 URLs. Additionally, we treated our evaluation as a "black box," with the ultimate concern being whether or not the malicious website was detected. A second difference between research environments and operational scenarios involves the features under study. Researchers often select features ahead time (*a priori*), based on the threat or based on what is known to be effective. This assumes that attack techniques do not evolve over time. We bridged a third gap between research and industry scenarios by limiting our features to those that could be gathered from a `response` to a web `request`. The benefit of using such features is that they can be gathered in the course of normal interactions with websites. Other research has used

additional features like domain name system (DNS) requests or search engine ranking, but this would require additional overhead and depends on those services being available.

1.2.5 Analysis on Different Datasets and Over Time

Finally, we focused this research on analyzing the applicability of findings from this study to other datasets and on conducting a study of feature-based malicious website detection over time. Researchers typically face the challenge of generating results that are specific to a study's individual dataset, which in this field often consists of gathering data applicable to a single threat and gathering it at a single point in time. To address that limitation, we explored whether and how the detection models and their features could be applied to other datasets. Additionally, we conducted research on an additional dataset that was gathered over time.

1.3 Research Questions and Approach

In conducting this research, we evaluated an approach for identifying features for malicious website detection in various scenarios and over time. We approached our work on the basis of the 13 research questions outlined in the following sections.

1.3.1 Research Question 1

With our first research question, we addressed how well our approach aligned with or diverged from prior research. In our survey of prior research on malicious website detection, we observed that several features were reused for malicious website detection, opening the opportunity to identify new features. We hypothesized that by considering additional features (many of which had never been studied for malicious website detection in the past), we would identify new features as being important to the detection of malicious websites. With RQ1, then, we investigated how the features identified

through our approach differed from those gathered from prior research. RQ1 is stated as follows:

RQ1: How do the features identified compare with prior research?

1.3.2 Research Question 2

We used our second research question to investigate whether the identification and incorporation of new features improves malicious website detection. Although we captured many performance metrics for the models we built, we focused our discussion and performance comparison on the Matthews Correlation Coefficient (MCC) [27] since it handles imbalanced datasets. To do so, we built detection models from features identified in our research and in prior research, comparing the respective MCCs from models built from features exclusively from prior research. We repeated this approach under two feature transformation scenarios – feature transformation with feature selection and feature transformation with PCA [28]. Hence, RQ2 is stated as follows:

RQ2: Do the additional features identified improve malicious website detection?

1.3.3 Research Question 3

Our third research question enabled us to examine the effect of dataset imbalance, a constraint that is common to malicious website detection experiments. The datasets used for malicious website detection experiments typically contain imbalance – an unequal number of malicious and benign websites. To investigate the effects of conducting experiments with an imbalanced dataset, we trained our models on different samplings of our training dataset. We then evaluated the models to determine the impact on overall detection performance. Hence, RQ3 is stated as follows:

RQ3: Do our results change with no-sampling, under-sampling, and over-sampling scenarios?

1.3.4 Research Question 4

Here, we aimed to compare the performance of features identified in our approach to the performance of those features identified in prior research. To do so consistently, we built all of the models with the default parameters provided by [29]. However, it was possible that we could obtain better results by performing hyperparameter tuning and cross-validation of our models. Therefore, we focused RQ4 on hyperparameter tuning and cross-validation of our models, stated as follows:

RQ4: Does hyperparameter tuning and cross-validation improve our results?

1.3.5 Research Question 5

We focused the fifth research question on the results of using all features in this study followed by feature selection to discover features for malicious website detection. Using the webpage content features, URL features, and Hypertext Transfer Protocol (HTTP) header features as the basis for the detection model provided the best understanding of how discovering features (versus selecting them ahead of time) performed. RQ5 is stated as follows:

RQ5: Is feature discovery feasible for malicious website detection?

1.3.6 Research Question 6

Even if the discovered features (those features identified through feature selection) performed well, we still would have little understanding as to whether it was worthwhile to re-select features or if those from prior research were sufficient. To address

that gap in understanding, we needed to provide a comparison. RQ6, then, is stated as follows.

RQ6: How do discovered features' detection ability compare to those from prior research?

1.3.7 Research Question 7

The features used in this research can all be derived from the `response` to a web `request`. As such, the features were available to a normal web browser or HTTP environment and did not require any additional resources for collection. Although this set of features was limited, it could be used to supplement any other that is available based on the specific operational scenario. Hence, we arrived at RQ7, stated as follows.

RQ7: Can a discovery approach be applied to several threats when only features from a web response are available?

1.3.8 Research Question 8

We leveraged three datasets in conducting this study. Prior research has demonstrated the difficulty of applying detection models built from one dataset to another. However, to verify or refute this observation from prior research, we explored RQ8, stated as follows:

RQ8: How robust are malicious website detection models when applied to a new dataset?

1.3.9 Research Question 9

The next area of focus – a follow-on to the previous research question – addressed whether the features identified throughout this research, not just the models built from

them, could be used to build detection models on another dataset. RQ9 is stated as follows:

RQ9: How do the features identified perform on a new dataset?

1.3.10 Research Question 10

In the next area of focus, we explored whether we could apply aspects from the previous research questions to the new dataset to improve malicious website detection. RQ 10 is stated as follows:

RQ10: What aspects from prior experiments can we apply to a new dataset?

1.3.11 Research Question 11

All prior research questions were explored in the context of two datasets, both of which were captured at a single point in time. At this point in the research, we shifted our approach, focusing the last three research questions on temporal aspects of malicious website detection. The first aspect of our temporal study included an evaluation of the how detection performance changes over time, with RQ11 stated as follows:

RQ11: How does detection performance change over time?

1.3.12 Research Question 12

The internet is dynamic, with websites commonly assumed to change over time. Prior research has demonstrated that the web changes, but this assumption must be revisited for the purpose of this dissertation. To seek a rationale for the results of the previous research question, we explored RQ12, stated as follows:

RQ12: Do websites change over time?

1.3.13 Research Question 13

Once we determined whether websites changed over time, we went on to explore the degree of change. We did so by comparing the change in features as a function of time (1 week, 2 weeks, … 11 weeks), gathered from several measurements. Research Question 13 is stated as follows:

RQ13: If websites change over time, how much do they change over time?

1.4 Contributions

Our contributions are listed below.

1. We identified new features for malicious website detection and validated the use of features from prior research in malicious website detection.
2. We quantified and compared the performance improvement when incorporating new features for malicious website detection.
3. We evaluated this approach on a dataset consisting of several types of malicious websites in order to demonstrate the approach's potential and explored additional datasets.
4. We evaluated and compared the performance of our detection method over several scenarios, varied the ratio of benign to malicious websites, used feature transformation, and performed hyperparameter tuning and cross-validation to explore consistency.
5. We demonstrated the feasibility of discovering features for malicious website detection and the advantages of doing so over choosing features *a priori*.
6. We quantified the performance of detection models over time and compared the degree of website change over time.

1.5 Dissertation Outline

Figure 1-4 shows a detailed overview of the structure of this dissertation. We have structured this dissertation in the following manner. In Chapter 2, we present a survey of related work on malicious website detection. Chapter 3 details our methodology. In Chapters 4–6, we describe the independent studies conducted on different types of features for malicious website detection, dividing the material with: the webpage content in Chapter 4, the structure of the website URL in Chapter 5, and the HTTP headers from the website in Chapter 6. In each of these chapters, we address research questions 1-4 and we outline the similar methods of feature selection, feature ranking, and model training and evaluation applied in each, with the main difference being the type of features studied. The works described in Chapters 4 and 6 have been published [30], [31] and have been presented at two conferences. In Chapter 7, we address research questions 5-7 and explore our use of all the features studied to that point – webpage content, URL structure, and HTTP headers for detection. Chapter 8 investigates research questions 8-10 and includes details regarding our application of models and features identified through this dissertation to a different dataset. In Chapter 9, we conclude with research questions 11-13 and outline the portion of the research aimed at determining whether and how the models for malicious website detection and the features for detection changed over time. We discuss limitations in Chapter 10. Finally, we present a summary of the research and findings in Chapter 11.

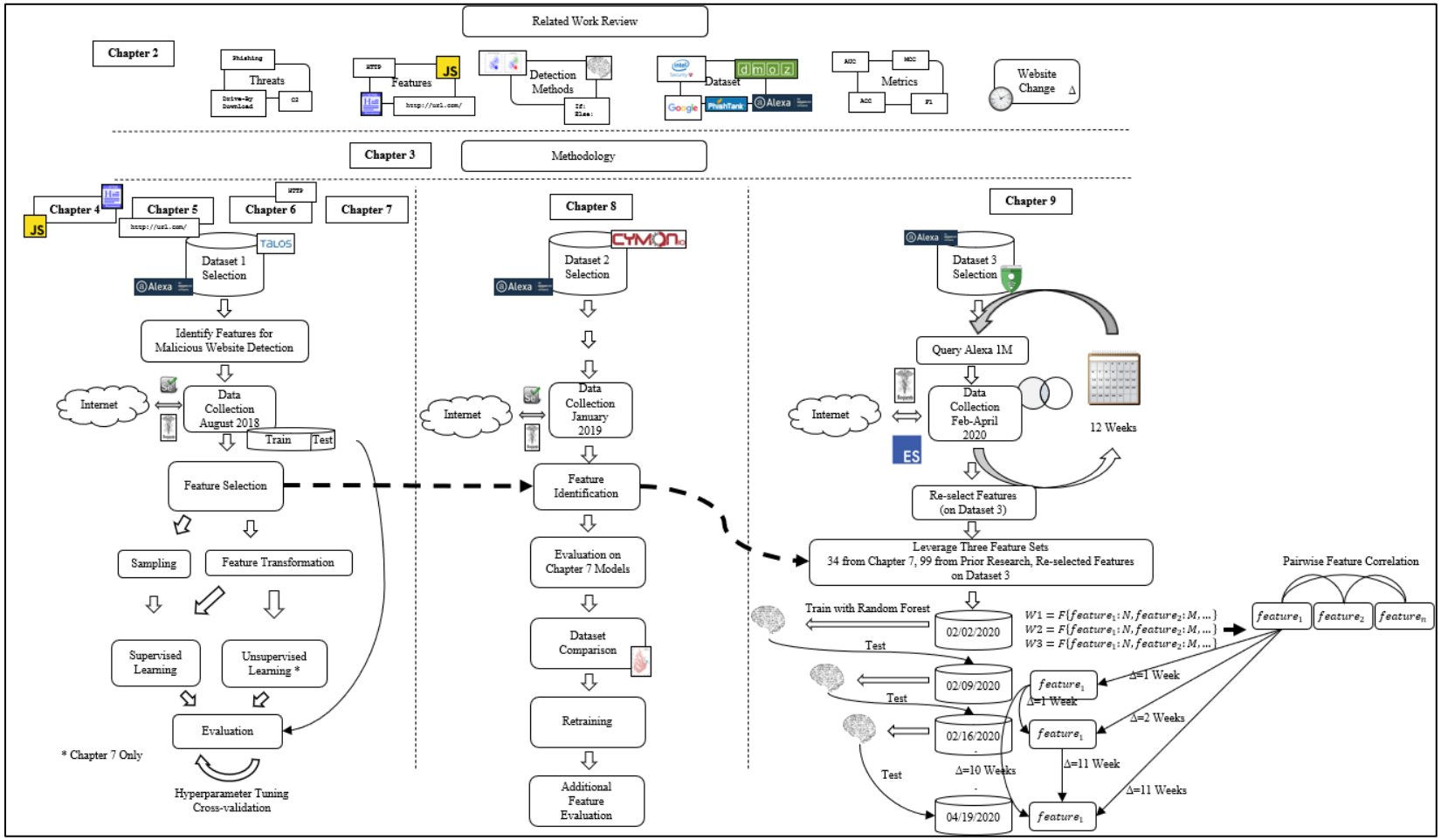


Fig. 1-4. A detailed overview of this dissertation
(Images courtesy of Pixabay [22].)

Chapter 2: Background and Related Research

2.1 Introduction

The techniques for detecting malicious websites have evolved over the years, as have the features used to detect them. The three facets common to all approaches to detecting malicious websites are: 1) the set of features that characterize a website; 2) the method(s) or model(s) used to make the determination; and 3) the dataset(s) used for training and validating the methods used to make the determination. In this chapter, we provide a survey of related research into each of these facets of malicious website detection. Additionally, we discuss four additional relevant aspects: 1) potential validation methods on an additional dataset; 2) potential practical implementations; 3) relevant performance metrics; and 4) measure of change in a website and training and evaluating detection models on different points in time.

2.2 An Overview of Features for Malicious Website Detection

The first aspect of malicious website detection is the set of features or quantifiable attributes that characterize a website. These features serve as the basis for determining whether a website is malicious. Researchers have drawn on a diverse set of features, including features in the following three categories: host information, webpage content, and communication data. The features in these categories include: the URL, the content of the webpage, network traffic to and from the website, information available in the DNS [32] and registration records, geographic properties, and certificate information [33].

2.2.1 Host Information

For our purposes, we define host information as being all aspects of a website that must be in place before the website is accessed. Examples of host information include the URL, information found in the domain name registration system, and the website certificate. In this section, we discuss the URL features that are the most prevalent host information features used in prior research for malicious website detection.

2.2.1.1 URL Word-Based Features

Word-based features are motivated by the observation that phishing URLs often contain specific words or can be tokenized based on specific delimiters for further analysis. One of the early word-based approaches in malicious website detection and classification came from [34], who discovered a list of words notably found in phishing website URLs. These words, which included “webscr,” “secure,” “banking,” “ebayisapi,” “account,” “confirm,” “login,” and “signin,” were used as a group of features to detect phishing URLs. The words “login” and “signin” were found to be particularly prominent on their phishing dataset. Ma et al. [35] expanded on this approach and implemented a method that separates the path in the URL by special delimiter characters (“/,” “?,” “.,” “=,” “-,” “ and “_”) into tokens for further analysis. This approach, referred to as a “bag of words” approach, is a common approach to URL feature generation. Ma et al. [36] repeated this approach with the addition of an online learning algorithm and continued the research [37]. The “bag of words” approach for phishing detection has also been used by other researchers [38]-[40] and is one of the primary methods for analyzing URLs. Word-based features and the “bag of words” approach can be used to detect all types of

malicious website URLs, but the approaches have been used predominantly to detect phishing URLs.

2.2.1.2 Special Characters and URL Structure Features

Researchers also have explored the use of special characters and the URL structure for detecting malicious websites and URLs. This differs slightly from using special characters as delimiters for the "bag of words" approach. One characteristic in the structure of the URL is the presence of an internet protocol (IP) address [41]. IP addresses can be substituted for hostnames and are sometimes used by malicious websites to hide malicious domain names for phishing, drive-by downloads, or C2 websites. Researchers [42] stated that IP addresses in URLs could be indicative of a malicious URL and used the presence of an IP address in the URL as a feature. In addition, they also counted the number of hosts in the URL that could be determined by counting the number of "." characters in the URL. The number of dots is motivated by an observation that malicious websites use multiple hostnames in order to appear more legitimate. He et al. [43] also considered the presence of the "@" character as a feature. Authors [44] reused the features mentioned thus far and added the presence of a "shifted" URL, multiple top-level domains (TLDs), misspelled domain names, modified URL encoding, and modified or mismatched port numbers, along with adding whether the URL was a short or a "tiny" URL. IP addresses, multiple hosts, having several TLDs in the URL, URL length features, and other special characters have all been used in some manner or permutation by researchers [40], [45]-[49] as features for detecting malicious URLs. Basnet et al. [50] used the presence of special characters as features and evaluated feature selection techniques on phishing datasets. The features mentioned in this section have

primarily been used to detect phishing websites, though some features – including the length of the URL, the number of dots (.) in the URL and ratios of characters to numbers – have proven successful in detecting other threats such as drive-by downloads and C2 or bot URLs.

Lin et al. [51] reused many features and presented an approach that used ratios within website URLs. Examples of ratios include: the length of the domain name divided by the length of the entire URL; the length of the path divided by the length of the URL; and the length of the argument field divided by the length of the URL. In addition to these ratios, [51] also used specific patterns such as letter-digit-letter and the longest word length as features. Ahluwalia et al. [52] focused on a specific type of threat – domains generated by a domain generation algorithm (DGA) [53] – and solely used URL length, number of vowels and consonants, and digits in the second level domain name to detect this specific type of malicious URL. The approaches based on ratios and the analysis of the distribution of vowels, consonants, and digits have primarily been leveraged to detect malicious URLs used by bots or C2 traffic with a detection and false positive rate (FPR) of 98.96% and 2.1%, respectively [52].

2.2.1.3 Additional Approaches with URL Features

Researchers [54] took an approach toward URL analysis that defined and used the Kolmogorov complexity of the URL string to identify malicious URLs. This approach did not require *a priori* knowledge and could be combined with other methods discussed in this section. Kheir et al. [55] classified C2 connections via statistical clustering of the URLs generated by a malware testbed. In [56], the authors used character n-grams from $N = 1$ to $N = 10$ appearing in the URL string to classify malicious URLs. Distinguishing

factors for [56] were their evaluation of the effectiveness of their n-gram approach on phishing as well as on spam URL datasets and their comparison of the respective performance on these datasets.

2.2.2 Webpage Content

Webpage content consists of the information gathered from the webpage that is available when navigating to the website URL. All webpage content features can be extracted from the webpage. This section includes a review of those features extracted from the webpage that are relevant for malicious website detection.

2.2.2.1 Term Frequency-Inverse Document Frequency (TF-IDF) and Its Applicability in Webpage Content

Term frequency-inverse document frequency (TF-IDF) is a statistical measure used to evaluate the importance of a specific word to a document [57]. It has been used in malicious website detection in several ways. The main methods involving TF-IDF are search engine and comparison based.

The authors of CANTINA [24] and CANTINA+ [46] extracted the top *K* words from a webpage and performed a Google search of those *K* terms. The authors then examined the top *N* returned results, with whether the webpage appeared in the top *N* results being used as a feature for malicious website detection. The researchers varied *K* (the number of terms) and *N* (the number of results), with that approach, along with the Google search engine, being used by [42]. Researchers [58] used TF-IDF to compare the contents of a candidate webpage with the contents of the TLD webpage of the candidate webpage. The larger the difference, the more likely it was that the candidate webpage was malicious. He et al. [43] used the difference between the candidate webpage and the

TLD page as part of a macro feature they referred to as “URL Identity.” In addition, the author of CANTINA+ [46] used TF-IDF to find the presence of specific sensitive words throughout the webpage. Researchers primarily used the TF-IDF approach, regardless of the specific implementation, as a means of identifying phishing websites.

2.2.2.2 Webpage Content - Structural Content - Tags and Attributes

HTML elements and attributes, as well as the document object model (DOM), are the defining portions of webpage structure and have served as a basis for multiple features for malicious website detection. One well-studied feature is the `<iframe>`. Provos et al. [59] studied the prevalence of drive-by downloads on the web and `<iframe>`s that are often used in malicious content injection common in drive-by downloads. Although `<iframe>`s facilitate content injection and drive-by downloads, other structural information can identify other attacks, such as phishing. Whitaker et al. [42] used a simple feature – whether the webpage has a password field – as one of many features to detect phishing webpages. Authors [43] expanded on this by extracting other features including `<meta>` tag and description tags, the `<title>` tag, and all text fields inside the `<body>` tag as feature sets to detect phishing websites. Other authors including [44], used similar features. Xiang et al. [46] captured the presence of “bad forms,” that is, forms with a specific structure (structures where the form was an HTML `<form>`, where keywords were related to sensitive information, and where there was a specific action attribute as a feature).

Basnet et al. [50] expanded the selection of structural features to include password fields, as well as counts of various tags within the webpage, including `<iframe>`s and `<frame>`s. Drew and Moore [60] extracted input HTML elements and performed

multiple stages of clustering to identify criminal websites that share commonalities. Corona et al. [61] took an overall approach similar to the TF-IDF difference between top webpage and candidate webpage. Instead of using terms, they used the difference in HTML between the candidate webpage and the TLD webpage. Borgolte et al. [62] aimed to detect malicious campaigns, extracted different features, and ignored visual differences.

Researchers also examined a group of features related to the URLs and to links present in a webpage. Several HTML elements have the `href` and `src` attributes, which specify links or references to additional resources such as files. The type of resource depends on which HTML element specifies the `href` or `src` attributes. The resources referenced by the `href` or `src` attributes can be on the webpage (like a section), or can be in another domain or website. Researchers [42] extracted features describing the extent to which links and images reference other domains outside of the TLD for that webpage. He et al. [43] did the same, extracting the base domain by extracting the `href` attributes from the `<a>` and `<area>` elements, while [48] only used the `<a>` tag in their identity builder. Gastellier-Prevost et al. [44] expanded their feature set to other HTML elements with the `href` attribute. With CANTINA+, the authors [46] checked whether the majority of URLs in the webpage were within the same domain as the candidate webpage. Le et al. [63] used the presence of external links in `<frame>` tags to capture a macro feature they called “foreign contents.” With BINSPECT, the authors [47] counted the total number of links and split them into categories similar to those created by other authors, including same-origin and different-origin. Eshete et al. [47] also counted the number of external JavaScript files in the URLs on the webpage. Like the researchers

who had examined URLs and their relationships to TLD webpages, [50] checked whether the respective `<iframe>` links pointed to internal or external resources or to other denylisted URLs and expanded on that research in a later study [64].

2.2.2.3 Webpage Content - Defining Page Content Behavior with JavaScript

Behavioral features of a website come primarily in the form of JavaScript, an object-oriented programming language and a foundational technology of the modern web. Because JavaScript is a powerful language that can be misused by attackers, it has been of interest to various researchers. JavaScript is most commonly misused to enable drive-by downloads. JavaScript Anomaly-based Analysis and Detection (JSAND) creators [65] focused on identifying malicious webpages with drive-by downloads by extracting JavaScript features, gathering features by executing the JavaScript in a sandbox and recording features during execution. Although the researchers collected several features, they focused their study primarily on the detection and execution of suspicious behavior, including suspicious methods and sequences of method calls, the presence of likely shellcode, and indicators of JavaScript obfuscation (a method used to hide malicious code from someone analyzing the script). Canfora and Corrado [66] also leveraged JavaScript features in research focused on the detection of malicious websites. The authors addressed features such as the presence of suspicious methods, specific sequences of method calls, and indicators of obfuscation. In addition, they compared groups of features in order to determine which features were best able to detect malicious websites. They found that JavaScript played a significant role in malicious website detection.

Other authors approached JavaScript analysis in malicious webpages from the abstract syntax tree (AST). Rieck et al. [67] proposed Cujo, which has both static and dynamic (execution) analysis components. Cujo is trained on reports detailing benign and malicious code, with its performance evaluated with either static features or dynamic features alone or with static and dynamic features combined. The authors found that using static and dynamic features together improved their accuracies compared to using static and dynamic features in isolation. Curtsinger et al. [68] used a mostly static JavaScript analysis approach, but made the argument that static analysis was a challenge for malicious JavaScript because malicious JavaScript is most likely obfuscated and hence is difficult to analyze statically. As a result, they hooked the JavaScript runtime to get the de-obfuscated JavaScript before analyzing the JavaScript AST statically. Researchers [69] used JStill to leverage the AST, but created four categories: 1) JavaScript native functions, 2) JavaScript built-in functions, 3) DOM methods [70] (those methods that operate on the DOM), and 4) user-defined functions that group their features. With JStill, the researchers captured three differences between malicious and benign method invocations: 1) the method arguments, 2) the method definition, and 3) the context of a method invocation. Kapravelos et al. [71], with Revolver, also used the AST with a focus on AST similarity between known malicious and candidate ASTs.

2.2.2.4 Combining Page Structure and Behavior for More Holistic Malicious Detection

Although structural features like TF-IDF, HTML, links, and URLs in the page, along with behavioral features like JavaScript, can be extracted independently to identify malicious websites, they are often combined. Researchers [23] described three categories

of features that combined structural and behavioral features. These features describe the exploit, the exploit delivery mechanism, and whether there are attempts to hide elements or scripts on the malicious webpage. The authors incorporated specific tags like `<frame>` and `<iframe>`, as well as indicators of JavaScript obfuscation, among their feature set. Choi et al. [72] also looked for the presence of suspicious native JavaScript methods like `escape()`, `eval()`, `link()`, `unescape()`, `exec()`, `link()`, and `search()`, combining them with HTML features including tag counts, and counts of zero size, and thus invisible, `<iframe>` tags. Heiderich et al. [73] proposed ICEShield, which lightly instruments JavaScript and detects attacks against the DOM tree. This approach combined attacks against the DOM with additional heuristics centered around previously studied HTML tags and considered the presence of suspicious Unicode as an additional feature. With Prophiler, [45] examined the `src` attributes of `<iframe>` tags, hidden elements, `<iframe>`s with small areas, and other features commonly found in malicious webpages. They also extracted 25 features around JavaScript code. With BINSPECT, the authors [47] extracted 25 webpage content features, primarily from prior research, including document length, number of words, lines, spaces, average word length, hidden elements, and presence of suspicious methods. In addition to capturing a better representation of the website, combination approaches are more applicable to detecting a wider range of attacks, as in the case of BINSPECT [47], which detected various malicious websites, including phishing and drive-by downloads, with 97% accuracy.

2.2.3 **Communication Data Features**

Communication data features describe the facet of the website that characterizes how a client communicates with the website. This includes protocol information, metadata from the communications, and traffic summary statistics.

2.2.3.1 Communication Data Features – HTTP Headers

HTTP [74] is the primary application level protocol used throughout the web. As such, HTTP features are studied and used to detect malicious websites. HTTP features are most commonly used to detect C2 traffic and HTTP `request`s generated from malware. Authors [75] and [76] clustered HTTP communications from known malware and generated signatures. Researchers [55] executed malware in a sandbox that generated HTTP communications and took a clustering approach to grouping URLs in the malware-generated HTTP traffic to classify the C2 communications.

Tao et al. [77] gathered HTTP header features from interaction with a webpage and recorded attributes from the HTTP `request`s and `response`s over a session to a candidate webpage. The authors combined these features with non-HTTP features to detect malicious websites. Zhang et al. [78] examined features over a session, but focused specifically on redirect chains (one or more redirects) between the initial URL and destination. Brezo et al. [79] proposed a method of detecting malicious web requests by using machine learning and HTTP and transmission control protocol (TCP) characteristics. Although they found TCP features, such as packet length, to be the most influential in their study, HTTP characteristics still were among the top 10 most relevant features. Xu et al. [49] used 15 HTTP header features in addition to taking a “crosslayer” approach that used application, network, and webpage level characteristics. Specifically,

they reused the HTTP header `content-length`, which also was used by [79]. Researchers [80] proposed ExecScent, which generated control protocol templates (CPTs) from clusters of HTTP `request`s associated with C2 traffic. CPTs are defined by the URL, HTTP headers, and the destination IP address. Researchers [40] and [81] used HTTP headers – including the `response` code, HTTP method, and Boolean values such as if the HTTP `response` content is zipped – in their feature set for malicious website identification. Zarras et al. [82] created a method that learned how HTTP based malware worked and learned the structure of the HTTP requests sent. They leveraged header chains and templates like CPTs to use header chains to detect C2 traffic. Researchers [83] used Phishmon to examine the headers and used the length of the respective header values as features to detect phishing webpages.

2.3 The Methods and Models for Detection

The next aspect of malicious website detection is the method or model used to make the determination of whether a website is malicious. The method or model uses features that characterize the website to make the determination. In our literature survey, we found three types of methods that researchers used for detecting malicious websites: 1) heuristics, 2) clustering, and 3) supervised machine learning techniques.

2.3.1 Heuristics

Heuristics are simple approaches or rules that have been applied to detecting malicious websites. Their use was more prevalent in earlier research. Recent research has tended to favor the use of machine learning techniques. The main benefit of heuristics is their simplicity and intuitiveness, though they rely strongly on preconceived notions of malicious behavior or attributes. Seifert et al. [23] presented one of the earlier approaches

that leveraged heuristics to identify malicious websites. Their approach used many features in HTTP `responses` and the webpage HTML. Prakash et al. [84] used denylists as the basis to detect phishing attacks. Researchers [44] defined 20 heuristics from lists and acceptlists and implemented them in an anti-phishing toolbar called Phishark to differentiate between legitimate and phishing websites. Wang et al. [85], with Phishnet, evaluated rule-based and classifier-based approaches for identifying webpages that lead to drive-by downloads. In their study, the rule-based system outperformed their classifier, further motivating the continued use of heuristics. Nguyen et al. [86] created a heuristic with weights for six features to detect phishing websites. Ghafir and Prenosil [87] extended this idea, using threat intelligence to automatically update their denylist, which was leveraged to identify C2 traffic based on denylists of malicious IPs. Seshagiri et al. [88] created heuristics for known attack patterns with JavaScript and HTML. Authors [89] created the Phidma algorithm consisting of five layers in a pipeline to identify the webpage as legitimate, the five layers being: 1) acceptlist, 2) page features, 3) search engine, 4) URL similarity, and 5) accessibility. Heuristics are still relevant; however, most researchers in the field of malicious website detection leveraged more sophisticated machine learning techniques.

2.3.2 **Clustering**

Clustering has an advantage over heuristics in that it does not require preconceived notions of what malicious looks like. Clustering groups similar data, but usually requires larger amounts of data to create more defined clusters. Clustering has been successful in identifying several threats including threats detected via the webpage and via HTTP traffic.

Borgolte et al. [62] searched for new web infection campaigns by looking at two versions of the webpage, extracting differences in their DOM and assigning the difference to specific clusters. Drew and Moore [60] identified criminal websites by clustering websites based on metrics gathered from the HTML on the page. Researchers [75], [76] performed coarse grained clustering which measures the statistical similarity of the HTTP `request`s including total number of `request`s, number of `GET` and `POST` `request`s, and fine-grained clustering, which considers the structural similarity of the HTTP communications generated from malware in their testbed to generate detection signatures. Authors [90] built CyberProbe, which probes different servers and builds signatures known as fingerprints by clustering `request-response` pairs (RRPs) in the generated traffic. Kheir et al. [55] presented Webvisor, which records HTTP `request`s from known families of malware and then performs clustering of the generated URLs to build signatures for C2 channels. Zarras et al. [82] used a dataset of 40,000 malicious HTTP `request`s from 24 malware families and `request`s to the top 1,000 domains from Alexa to generate 7,000,000 HTTP `request`s and built HTTP templates from clustered HTTP headers.

2.3.3 **Supervised Learning**

The most common method of detecting malicious websites is to build models using supervised machine learning techniques. The features are extracted from known benign and known malicious websites to build models using one or more supervised learning algorithms. Some researchers [40],[42]-[43],[61] and [91] used one classification algorithm. This approach has shown success, with [91] being able to classify phishing webpages written in English with an area under the curve (AUC) of 0.999. Authors [42]

focused on classifying a “large” number of phishing webpages and training their logistic regression (LR) classifier [92] on millions of webpages. They evaluated their classifier on 165,382 phishing webpages during the first six months of their study. Authors [43], [61] used a support vector machine (SVM) [93] classifier with a different set of features and, unlike [42], used a smaller evaluation dataset of 200 legitimate and 325 phishing webpages in their experiment. Authors [40], [43] also leveraged an SVM-based classifier while [91] used a gradient-boosting (GB) classifier [94].

Other scholars [35],[45]-[47],[49]-[50],[72],[83], [95] used up to seven different algorithms. Ma et al. [35] leveraged an LR classifier (a naïve Bayes [96] and SVM-based classifier) and also recorded the time to test and train their classifiers. Canali et al. [45] used random tree [97], random forest (RF) [98], naïve Bayes, LR, J48 [99], and Bayesian networks [100] and compared their respective performances. RF was the best performing classifier over different feature sets. Similarly, [46] used Bayesian networks, J48, RF, AdaBoost (AB) [101], LR, and SVMs. Choi et al. [72] included Ra*k*El [102] and multi-level K-nearest neighbor (ML-KNN) [103]. Basnet et al. [50] used seven supervised classifiers and then combined them with a customized version of the confidence-weighted, majority-vote algorithm [104]. In [50], the authors used naïve Bayes, RFs, and LR classifiers. Researchers [49] performed a similar study with four classifiers and found J48 to be the best performing. Researchers [95] also demonstrated applicability of decision tree classifiers, particularly J48. Phishmon creators [83] used similar algorithms, but added classification and regression trees [105] into their study.

Authors [106] used batch learning, where models are built on the whole dataset at once, while others [107] used online learning, where models were updated as data was

made available. Ma et al. [36] continued their work from [35], but with online learning algorithms including Perceptron [108], LR, Passive-Aggressive algorithm [109], and confidence-weighted algorithm. Both [39] and [64] used batch learning and online learning as well. Other authors, including [48], applied more than one classifier, usually in sequence, in their detection schemes. Several algorithms were used, including AdaBoost, Bayesian networks, CART, confidence-weighted, C4.5 [110], GB, J48, K-nearest neighbor (KNN) [111], LR, naïve Bayes, Perceptron, RF, random tree, and SVMs.

2.4 **Validation**

Ground truth datasets used for training and evaluation make up the next component of malicious website detection. Currently, no standard dataset exists for training machine-learning algorithms to detect malicious websites, though some datasets have been reused by several researchers. We identified three types of datasets used in malicious website detection: 1) well-known datasets, 2) custom datasets, and 3) proprietary datasets provided by an external organization. Well-known datasets are commonly used as ground truth for malicious and benign websites. Examples of such datasets include Alexa.com [112] for benign domains or Phishtank [113] for malicious domains (in the case of phishing related studies). Multiple researchers [24], [44], [47]-[49], [61], [73], [77], [86], [89], [114]-[117] used these predefined datasets. Researchers who used the second type of dataset – a custom dataset, commonly generated "randomly" or by a crawler – include [23], [35]-[37], [40], [42]-[46], [49]-[50], [62], [64], [66]-[67], [69], [72], [81], [91], [95], and [118]-[119]. Although the random and crawler-based approach can be used for both benign and malicious dataset generation, it has been more

commonly used to generate data for benign websites. Moreover, this type of method can be combined with well-known datasets. The third type of dataset is a dataset provided by external organizations. These datasets, used by [55], [91], [119]-[120], are not as common. The nuances and differences among datasets used by prior researchers can be subtle. As a result, we created Table 2-1 below to briefly describe these works and their benign and malicious datasets. A value of "-" indicates that the specific field was not applicable in the respective study or that the author used a custom dataset specific to the study. A "*" character indicates that some or all of the data was provided by an unspecified external organization.

Table 2-1.
Datasets from Prior Research (Prior Research Leveraged Various Datasets Derived from Numerous Sources)

Datasets from Prior Research			
Work	***Year***	***Benign Dataset Source***	***Malicious Dataset Source***
[114]	2006	-	[121]-[122]
[24]	2007	[123]	[113]
[23]	2008	[124]-[125]	[126]
[35]	2009	[124]-[125]	[113],[127]
[36]	2009	[125]	*
[77]	2010	[112]	[128]-[130]
[42]	2010	-	[131], *
[67]	2010	[112],[132]	[133]
[116]	2011	[112], [121],[123]	[113],[122]
[44]	2011	[112], [135]-[136] *, -	[113],[137]
[45]	2011	[112], [132]	[133]
[63]	2011	[125], [134]	[130],[138]-[139]
[37]	2011	[125]	*
[46]	2011	[124]-[125]	[113], [141]-[140]
[47]	2012	[112], [124]-[125]	[113],[132],[142]
[50]	2012	[124]-[125]	[113]
[86]	2013	[124]	[113]
[49]	2013	[112]	[130], [143]-[146]
[48]	2013	-	[113]
[62]	2013	-	[133]
[64]	2014	[124]-[125]	[113]
[69]	2014	[112]	[147]
[40]	2014	-	-
[117]	2014	[124]	[113]
[118]	2015	[112]	[147]-[150]
[120]	2015	[151]	[151]
[55]	2015	-	[147], [152]
[115]	2016	[112]	[113]
[66]	2016	-	[153]
[91]	2016	[154]	[113]
[95]	2016	[124]	[113], [155]
[89]	2017	[156]	[113]
[61]	2017	-	[113]
[116]	2018	[157]	[113]
[119]	2018	-	[158]

Some prior researchers performed analysis on datasets derived from different sources. These datasets can vary temporally (identified in Table 2-2 below as “temporal”) or can be drawn from a different corpus (identified in Table 2-2 below as “corpus”). Table 2-2 provides a summary of the works of researchers who performed analysis on different datasets and how they differed, with “Y” meaning “yes” and “N” meaning “no.”

Table 2-2.
Prior Research Occasionally
Tested Detection Methods on
Different Datasets

Application to Another Dataset		
Research	***Corpus***	***Temporal***
[35]	Y	N
[38]	Y	N
[42]	N	Y
[43]	Y	N
[45]	N	Y
[46]	Y	N
[50]	N	Y
[63]	Y	N
[64]	Y	N
[65]	Y	N
[82]	N	Y
[159]	Y	N
[160]	Y	N
[161]	Y	N

2.5 **Practical Implementation**

An additional component, often specified in the related research, is the incorporation of detection models into a practical solution. Rieck et al. [162] tested Botzilla on a live university network by incorporating their approach in the open-source flow monitor Vermont [163]. They deployed their solution at the central gateway of a university network to monitor uplink traffic. Given high network traffic volume, they

only monitored the first 256 bytes of each flow to keep stream reassembly to a minimum. Cujo, developed by [67], was embedded in a web proxy between the web client and the web service. Cujo performed the analysis before data were sent to the web client and webpages containing drive-by downloads were blocked. Authors [164] divided their solution for detecting clickjacking attacks into two components – a detection unit and testing unit. The detection unit combined two browser plugins and the testing unit was a single browser plugin. Gastellier-Prevost et al. [44] took a similar approach, implementing the Phishark toolbar as a Firefox add-in. Ghafir and Prenosil [87] leveraged additional servers to passively analyze network traffic looking for denylist hits. Their approach also updated their denylist from various intelligence feeds. DeltaPhish was wrapped inside a web application firewall that served as proxy between the user and the website in Corona et al.'s [61] live implementation.

2.6 Performance Metrics

Performance metrics include the FPR, false negative rate (FNR), true positive rate (TPR), true negative rate (TNR), accuracy (ACC), AUC, Precision (Prec), Recall (Rec), F Score, and MCC. In some cases, authors specified other metrics such as detection and error rate. Currently there is no standard set of metrics used for evaluation that is consistent across malicious website detection studies. To better understand the capability of prior approaches, we listed related research, the relevant performance metrics, and the results. Table 2-3 below lists those works that were most similar to our research, as well as those that provided concrete numbers (as opposed to graphs and visualizations alone). The table also includes the results from the respective research that we identified as the "best" or as capturing the most "representative" reflection of their approach. Many of the

works provided several measurements with slightly different features and datasets and quantified additional performance aspects like time. Therefore, the selection of the “best” or most “representative” result was somewhat subjective. A value of “-“ indicates that the respective metric was not discussed in the respective related research.

Table 2-3.
Prior Researchers Did Not Use a Standard Performance Metric

Performance Metrics from Related Research											
Related Research	Year	TPR	TNR	FPR	FNR	ACC	AUC	Prec.	Rec	F Score	MCC
[24]	2007	0.97	-	0.06	-	-	-	-	-	-	-
[23]	2008	-	-	0.0588	0.4615	-	-	-	-	-	-
[35]	2009	-	-	-	-	0.99	-	-	-	-	-
[77]	2010	-	-	0.001	-	0.922	-	-	-	-	-
[67]	2010	0.944	-	0.00002	-	-	-	-	-	-	-
[65]	2010	-		-	0.002	-	-	-	-	-	-
[42]	2010	-	-	0.0003	-	-	-	0.9754	0.9497	-	-
[45]	2011	-	-	0.0988	0.0077	-	-	-	-	-	-
[43]	2011	0.9733	-	0.0145	-	-	-	-	-	-	-
[46]	2011	0.9424	-	0.01948	-	-	-	-	-	0.9607	-
[37]	2011	-	-	0.0152	0.0255	-	-	-	-	-	-
[44]	2011	-	-	-	-	-	-	-	-	-	-
[47]	2012	-	-	0.189	0.011	0.97	-	-	-	-	-
[54]	2012	0.969	0.9315	0.071	0.031	-	-	-	-	-	-
[48]	2013	0.969	-	0.0125	-	-	-	-	-	-	-
[49]	2013	-	-	0.03676	0.09127	0.95161	-	-	-	-	-
[86]	2013	-	-	-	-	0.97	-	-	-	-	-
[165]	2013	-	-	0.081	0.017	0.965	-	-	-	-	-
[69]	2013	-	-	0.0175	0.0053	-	-	-	-	-	-
[40]	2014	-	-	0.063	0.076	-	-	0.935	0.924	0.93	-
[117]	2014	-	-	0.013	-	0.995	-	-	-	-	-
[159]	2014	-	-	0.002	0.005	-	-	-	-	-	-
[64]	2014	-	-	0.0024	0.0075	-	-	0.9955	0.9925	0.994	0.991
[118]	2015	-	-	0.00212	0.008492	-	-	-	-	-	-
[81]	2015	-	-	-	-	-	-	0.935	0.924	0.93	-
[66]	2016	-	-	-	-	-	0.891	0.819	0.819	0.819	
[91]	2016			0.0005			0.999	0.956	0.958	0.957	-
[95]	2016	-	-	0.177	0.022	0.939	-		-	-	-
[52]	2017	-		0.021	-	-	-	-	-	-	-
[89]	2017	0.9054	0.9418	0.0582	0.0946	0.9272	-	-	-	-	-
[56]	2017	-	-	-	-	0.9848	-	-	-	-	-
[83]	2018	-		0.013	-	0.954	-	-	-	-	-
[166]	2018	-	-	-	-	0.964	-	0.964	0.964	0.963	-
[116]	2018	-	-	0.004	-	-	-	0.997	-	-	-

2.7 Measuring Website Change

For the portion of the research detailed in the ninth chapter of this dissertation, we measured the change in a website. It is often assumed that websites change, and prior researchers have quantified and measured such change. Researchers [167], motivated by the potential benefits of using caching servers, conducted one of the earliest studies in measuring change on the web. They found that only 22% of the web resources referenced in their traffic dataset were accessed more than once, with half of the 22% being accessed from multiple reference sources. In addition, they studied other changes on the webpage, including changes in `href`s (hyperlinks), images, email address, telephone number, and URL strings in the body of the webpage. Cho and Garcia-Molina [168] instrumented a crawler and crawled more than 700,000 pages, capturing whether a webpage changed (based on the MD5sum of the webpage). They reported that 40% of all webpages in their evaluation dataset changed in less than a week, breaking down which webpages changed based on the domain (.com, .netorg, .edu, and .gov). Fetterly et al. [168] expanded on this work by monitoring changes in other aspects of the website, including the webpage length and HTTP `response` code. Fetterly expanded on [170] in [171], shifting focus to determine how many webpages were duplicates and finding that 29.2% were very similar to other webpages and that 22.2% were near-identical. Brewington and Cybenko [172] monitored change and the lifetime of the webpages to model and infer change rates.

Lim et al. [173] measured frequency of web document change over time but did so on a "word" level. Ada et al. [174] examined webpage changes at a finer level than previous work by developing an algorithm that tracked the movement of DOM elements within the documents and evaluated the persistence of structural elements. Kwon et al.

[175] proposed criteria and a new metric for measuring webpage change based on six types of changes associated with webpages: “add,” “drop,” “copy,” “shrink,” “replace,” and “move.”

Although past researchers have emphasized the broad study of how websites change, we have not identified any metrics useful for our purpose of evaluating detection models over time. Although these studies have established that websites change over time, we revisited this assumption in Chapter 9.

2.8 Summary

From this literature review, we identified three common facets of malicious website detection: 1) the features used, 2) the method(s) or model(s) used to make the determination, and 3) the dataset(s) for training and evaluation of the models. We summarized studies that were performed on different datasets, identified prior research that incorporated research methods into practical solutions, and discussed performance metrics used in the prior studies. Additionally, we discussed research that measured websites and their change over time.

Chapter 3: Methodology

3.1 Overview

In our research, we evaluated a method for detecting malicious websites, leveraging features proposed in prior research, and also identifying new relevant features through statistical analysis. This method can be repeated to adapt with the evolution of threats and malicious techniques. Ultimately, we envision the repetition of this method over time, in order to identify sets of features and evaluate them for their applicability in detecting malicious websites. Although we focused on identifying and evaluating new features for malicious website detection and did not develop a new tool, the features that were identified and evaluated can be used as part of an additional layer of protection that can be hosted in a browser. Figure 3-1 below shows a potential use case wherein a detector built from a model using our method could examine and adjudicate the webpage before rendering it in a user's browser. Images courtesy of [22]. Our research consisted of several steps leading to malicious website models that can be placed in a user's browser environment as shown in Figure 3-1.

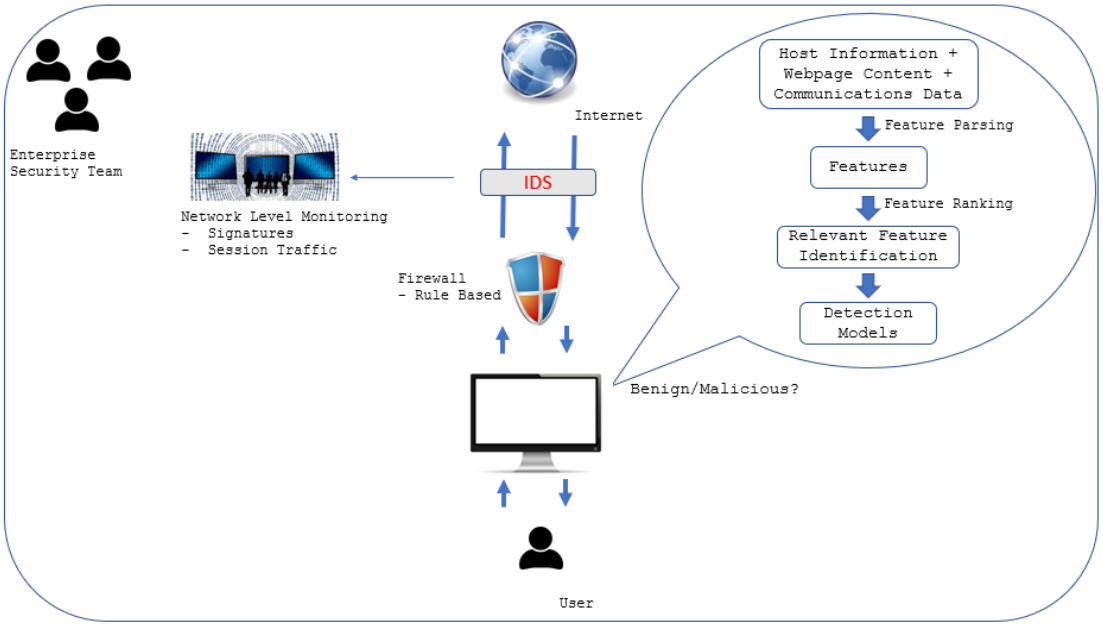


Fig. 3-1. Methods explored in this research can be applied with other security solutions

3.1.1 High Level Approach

We sought to identify and evaluate features for malicious website detection and evaluate them over time. We also compared the features identified by our approach to those identified in prior research in terms of rank and importance and in terms of the ability of the detection models they yield. We evaluated this approach over various scenarios, datasets, and over time. At a high level, we followed the overall approach outlined below. Steps 1-4 correspond to Chapters 4-6. Step 5 corresponds to Chapter 7. Step 6 corresponds to Chapter 8 and step 7 corresponds to Chapter 9.

1. Step 1: Select datasets
    a. Choose malicious and benign datasets for the research.
        i. There are 3 datasets (Dataset 1, Dataset 2, and Dataset 3).
2. Step 2: Discover features

a. Identify potential features from prior research that represent the three facets that characterize a website (host information, webpage content, and communication data).
b. Expand on features from prior research and incorporate new, unstudied features.
c. Select features for malicious website detection.

3. Step 3: Build detection models
    a. Individually evaluate feature-based malicious website detection using features from three facets that characterize a website (host information, webpage content, and communication data) by building detection models from supervised machine learning techniques over three scenarios. Scenarios include no-sampling, over-sampling, and under-sampling of the dataset to account for class imbalances between our malicious and non-malicious datasets.
    b. Rank the importance of features with regard to their ability to detect malicious websites.
    c. Apply pair-wise feature transformation techniques to identify additional features, followed by feature selection and PCA, and rebuild the models to further investigate the consistency of our approach over multiple scenarios.
    d. For training and evaluation of the models, use an 80:20 split of training to testing data, that is: 80% of the data is used to build the models, with the remaining 20% being used to evaluate the models.

    e. When applicable, compare the performance of models built with the features identified with the performance of models built with features from prior research.

4. Step 4: Tune and cross-validate
    a. Perform hyperparameter tuning and cross-validation of the detection models in an attempt to improve performance and demonstrate consistency in our models' detection ability.
    b. Repeat steps 3a, 3b, 3c, and 4a on a training-to-testing split of 70:30 to demonstrate that our results are not a product of the initial 80:20 training-to-testing split from step 3d.
5. Step 5: Combine features for improved detection
    a. Repeat steps 2 through 4, but use all of the categories of features to achieve better detection.
    b. Compare results to features used in prior research.
6. Step 6: Evaluate on another dataset
    a. Apply the RF model built in Chapter 7 to a new dataset (Dataset 2).
    b. Capture performance metrics on the model.
    c. Retrain a new model on the new Dataset 2, with features identified from Dataset 1.
    d. Investigate incorporating data from both datasets in training and evaluation.
7. Step 7: Explore detection performance over time

a. Measure the performance of a model trained on Dataset 1 and evaluated on another dataset (Dataset 3).
b. Investigate the impact of model re-training on performance, using:
    i. various feature sets, and
    ii. different training intervals.
c. Evaluate website change over time.
    i. Quantify the number of features (and their importance) relevant to malicious website detection change over time with statistical tests.

3.2 Step 1: Select Datasets

3.2.1 Dataset 1

There are many methods for choosing a benign dataset, but there are two popular paradigms – either create a new dataset or leverage existing datasets. Creating a new dataset has the advantage of enabling the researcher to select websites deemed representative of the websites on the internet or websites that are more relevant to the research topic. The disadvantage of creating a new dataset, on the other hand, is that it requires building a method of gathering relevant websites, such as a crawler, which could influence or sway results. Researchers can attempt to minimize influence on their dataset selection, but a practical way to remove researcher influence is to use a dataset provided by an external party. In our research, we chose a well-known and commonly used benign dataset source, the Alexa top one million domains (Alexa Top 1M) provided by [176]. At least 10 studies on malicious website decision used the Alexa Top 1M domains as a source of benign websites. In addition to being used as the foundation of the benign

dataset in prior research, we performed an additional check with threat intelligence information to ensure that the respective domains found in the Alexa Top 1M were not commonly involved in attacks.

Like the benign website dataset, malicious website datasets typically come from two places – a custom or an existing dataset. Just as we chose an existing dataset (the Alexa Top 1M) for the benign websites studied, we chose to use a dataset provided by an external party – Cisco Talos [177] – for the malicious websites. The dataset consists of malicious websites representing several classes of attacks including drive-by downloads, phishing websites, and C2 website URLs. By using the dataset provided by Cisco Talos, we lessened our influence over the dataset. Specifically, we did not choose the actual entries in the list. Additionally, the malicious dataset provided by Cisco Talos allowed us to evaluate our approach on an aggregation of websites associated with several types of threats and, therefore, attacks. This contrasts with previous researchers, who typically focused on a single type of attack or had *a priori* knowledge into exactly which types of threats were present in their malicious dataset. The combination of the benign and malicious portions of this dataset are referred to as Dataset 1. It is used primarily in Chapters 4–7 and is leveraged minimally in Chapters 8 and 9. Dataset 1 was collected in August 2018.

3.2.2 **Dataset 2**

We derived Dataset 2 from the websites in the Alexa Top 1M collected in January 2019. Once the collection was complete, we labeled the data using open source threat intelligence information provided by Cymon.io [193]. That is, we labeled entries that were present in the Cymon.io data as malicious and labeled entries that were not present

in the Cymon.io data as benign. Like Dataset 1, Dataset 2 can be viewed as being provided from an external source, with the choice of entries being outside of our control. We used Dataset 2 in the portion of the research outlined in Chapter 8.

3.2.3 Dataset 3

We collected Dataset 3 over a period of 12 weeks, beginning February 2nd, 2020 and ending April 19th, 2020. We derived this dataset from the Alexa Top 1M as well. On February 2nd, 2020, we conducted a query through Censys [176] to determine the Alexa Top 1M. After doing so, we began collection of these websites over the following week. On each subsequent week (February 9th, February 16th, February 23rd, etc.), we repeated the query to Censys, beginning the last collection on April 19th, 2020. We limited analysis to the entries that were consistent throughout each week – a total of 106,776 websites (106,776 websites were present in the Alexa Top 1M on the query conducted each week on February 2nd, February 9th, … and April 19th). We used Google Safe Browsing [132] as the source of ground truth and labeled our data based on the rating provided by this service. This dataset, consisting of 106,766 websites that were consistently present in the Alexa Top 1M for a period of 12 weeks and were labeled based on Google Safe Browsing, is referred to as Dataset 3 and is used in Chapter 9.

3.3 Step 2: Discover Features

The next step in our research centered on the discovery of which features to use to detect malicious websites. We surveyed academic and industry papers to determine the features that were commonly used to detect malicious websites. Although each researcher approached the detection of malicious websites in a slightly different way, we were able to summarize the types of features collected into three facets depicted in Figure 3-2: 1)

host information, 2) webpage content, and 3) communication data. These three facets describe a website, with each facet consisting of at least one category of feature. For example, WHOIS information and website URL structure are both categories of features in the host information facet. Images courtesy of [22].

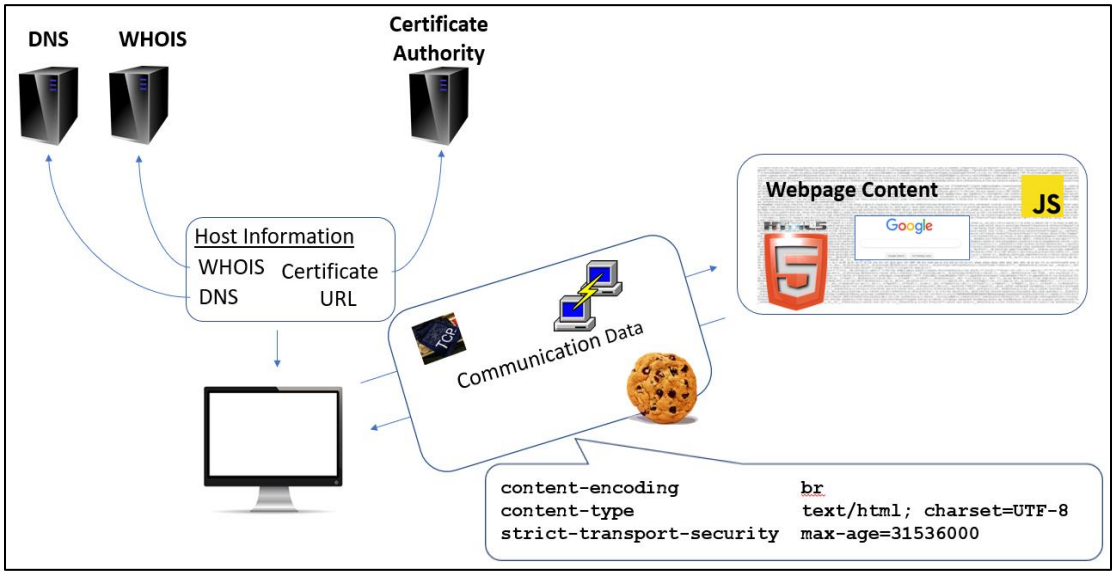


Fig. 3-2. Defining a website with three facets

Host information: We define host information as all information that must be in place before the website is accessed, e.g., URL, DNS information, and the presence of SSL certificates.

Webpage content: Webpage content consists of information gathered from the webpage fetched from the website URL. It includes the HTML, JavaScript, and any other information present on the webpage. Unlike host information, webpage content can be swapped out (in the case of updated HTML pages) without having to re-register the domain, URL, or any other corresponding information.

Communication data: We define communication data as information flowing to and or from the website. It can contain details at the traffic, protocol, or metadata levels

and governs the method or way to communicate with the website, e.g. HTTP headers, traffic statistics, and summaries of traffic flow over a period of time.

There are many categories of features that comprise these three facets, including: the structure of the URL, DNS record information, registration information, HTML and JavaScript characteristics, information gathered from TCP sessions, and HTTP metadata. Although many features exist, we limited our study of features to those that we could extract, just as a browser retrieves a website. Doing so reduced overhead during feature collection and increased the feasibility of this approach being integrated into a browser or other web client. Specifically, we used the URL structure as our host information features, we used the HTML and JavaScript on the webpage as our webpage content features, and we used the HTTP headers from the website as our communication data features. Refer to Appendices A, B, and C for a full list of features studied in this research.

3.3.1 **Extensive Feature Consideration**

Although prior research efforts identified features for detecting malicious websites, the researchers often relied on preconceived notions of the features to use. In some cases, these features have not changed over the years. For example, [23] counted the number of `<iframe>` elements on a webpage in their study and additional authors included this feature as well. Although [23] conducted their research more than 11 years ago, they helped establish the use of `<iframe>` information for detecting malicious websites. This single example illustrates the tendency of researchers to assume the relevance of certain features, based on their prevalence in prior research. In our research,

we included features gathered from previous studies, but also incorporated additional features and used additional techniques to determine which were useful.

3.3.2 Feature Selection Process

By gathering an extensive number of features, we ran the risk of overfitting our models, which would inhibit detection capability on unknown datasets. Also, using too many features could negatively impact computation performance for detection model building and evaluation. Thus, making detection decisions with hundreds or thousands of features was impractical. We sought, therefore, to identify a smaller set of relevant features for potential incorporation into a detector. To find such a set of features, we performed a series of feature selection steps after completing our feature collection, thereby identifying relevant features. We performed the six steps listed below to select features from our feature set.

1. Remove features for which all the features have the same value.
2. Remove features that have the same value at least 95% of the time.
3. Determine the variance inflation factor (VIF) [178], which measures multicollinearity (high correlations among independent variables), values for each feature and iteratively identify features that have a VIF > 5 [179].
4. Determine which features have similar VIF values and high correlation to each other (we defined high correlation as having a correlation coefficient greater than 0.7 [180]).
5. Iteratively repeat Step 4 and remove the highly correlated feature with the higher VIF from our feature set.

6. At this point, if our feature set consists of 50 or fewer features, we have arrived at our final feature set. If there are more than 50 features, however, we eliminate features even further with the use of XGBoost (XGB) [181], a GB algorithm. First, we calculate the feature importance – a metric between 0 and 1 that measures how much that feature impacts the algorithm's ability to make a determination regarding whether or not a website is malicious. We then iteratively input each feature importance as a threshold to the SelectFromModel technique [29], which is a transformer used to select features based on their weights. This produces sets of features that have a size "n" and corresponding threshold "t." With each set of "n" features associated with each threshold "t," we rebuild our XGB models to obtain an ACC for each set of 'n' features. We then iterate through the list of sets with "n" (the number of features) decreasing and identify relative maxima in the respective accuracies produced by the set of "n" features. When a relative maximum in ACC is observed, we stop and use the associated feature set as our final set. An example of this is seen in Chapter 4, Section 4.6.1. In Chapter 7 and 9, we observed that performing this step could directly (without Steps 3–6) produce a set of features for detection; hence, we skipped Steps 3–6 when performing feature selection in Chapters 7 and 9.

3.4 Step 3: Build Detection Models

3.4.1 Supervised Machine Learning Techniques

We used supervised machine learning classifiers to build our models. In its simplest form, supervised learning involves mapping input variables to an output variable

via a mapping function. However, the mapping function is learned from an algorithm that requires a known or labeled dataset as input. In our work, we had access to a corpus of labeled training data, both malicious and benign websites, which made a supervised learning approach feasible. The supervised classifiers used to build our models belong to several classes of machine learning algorithms: nearest neighbors [111], generalized linear models (GLMs) [92], ensemble methods [182], and neural networks (NNs) [183]. Among the models, four are ensemble methods and provide a measure of feature importance: adaptive boosting (AB), extra trees (ET), RF, and GB. The other models do not provide a measure of feature importance, but represent other classes of algorithms: bagging classifier (BC) [184] is an ensemble method [182], LR is a GLM, and KNN is a nearest neighbor [111] method. Covering additional classes of algorithms (other than those that incorporate feature importance), provides better insight into the effectiveness of the features identified and demonstrates consistency across various learning algorithms.

In this research, we leveraged: FPR, FNR, ACC, AUC, MCC, Prec, and Rec. We chose these metrics based on our motivation to present thorough and transparent results, on the prevalence of the metrics in previous research, and on the ability of the metrics to describe the detection ability of our models based in various ways.

1. Accuracy (ACC)

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}$$

where TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives, and FN is the number of false negatives.

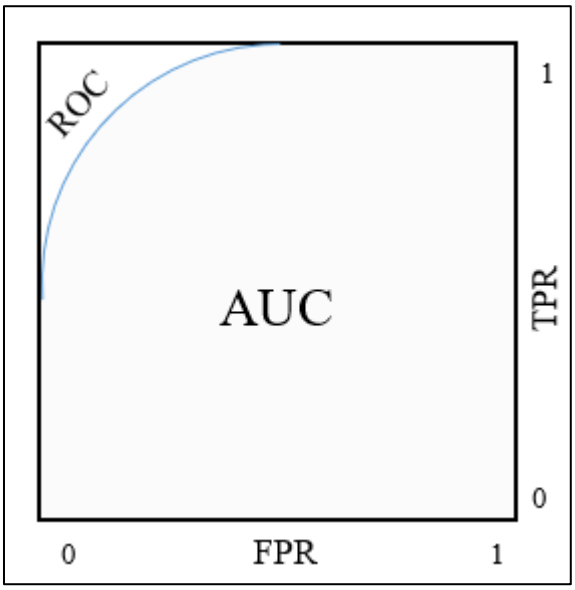
2. False Positive Rate (FPR)

$$FPR = \frac{FP}{TN + FP}$$

3. False Negative Rate (FNR)

$$FNR = \frac{FN}{TP + FN}$$

4. AUC is the area under the receiver operating characteristic (ROC) curve that plots the TPR vs FPR at each classification threshold. The AUC (lightly shaded) for a given ROC curve is shown below.



5. MCC is a measure of the quality of a binary classifier and ranges between -1 and 1, with 1 representing a perfect classifier, 0 representing a random classifier, and -1 indicating complete disagreement between the prediction and actual value.

$$MCC = \frac{(TP \cdot TN) - (FP \cdot FN)}{\sqrt{(TP + FP) \cdot (TP + FN) \cdot (TN + FP) \cdot (TN + FN)}}$$

6. Precision quantifies the number of correct positive classifications made and is defined as follows:

$$Precision = \frac{TP}{TP + TN}$$

7. Recall is a metric that describes how many positive cases the model finds from among all of the positive cases and is defined as follows:

$$Recall = \frac{TP}{TP + FN}$$

We focused our discussion in this research on the MCC since it incorporated the number of true positives and true negatives as well as false positives and false negatives in its value.

3.4.2 Importance Determination

Determining the most useful features for malicious website detection is a key task for building models that are not overfit and can be applied in a practical setting. To do so, we needed a method of ranking the importance of the potential features for malicious website detection. Fortunately, machine learning techniques such as AB, ET, RF, XGB, and GB algorithms can be used to build detection models and have the ability to provide feature rankings. Supervised machine learning techniques – including ensemble methods [182] and decision trees – have shown promise in prior studies. More importantly, each of these models provides a feature importance metric – a number between 0 and 1 that indicates how much the feature contributed to the model's classification decision. This importance metric allowed us to determine which features contributed the most to malicious website detection and to create a ranking of features. The sum of these feature importance metrics equals 1. A feature was considered more important (and higher ranked) than another feature if it had a higher importance.

Specifically, the models that calculate feature importance use decision trees and the Gini impurity [185] as the basis to determine feature importance. It is the foundation

for measuring feature importance and is calculated in a two-step process. First, we determine the Gini impurity [185] for a specific feature branch in the decision tree:

$$i(t) = 1 - \sum_{j=1}^{k} p^2(j|t)$$

where i(t) is the Gini impurity for the feature branch in the decision tree, t is the branch condition for the feature, k is the number of possible output categories (in our case k = 2 for malicious and not malicious), and p is the probability of each outcome in k given t. The total Gini impurity of that feature is created by taking a weighted sum of the respective indices per feature branch:

$$G(f) = \sum_{t=1}^{N} i(t) * p(t)$$

where G(f) is the total Gini impurity for a feature f, i(t) is the impurity of the respective branch, N is the total number of branches, and p(t) is the probability of that condition over the total dataset. The lower the Gini impurity, the more useful (important) the feature is in the decision tree and the higher it should be placed in the tree. Specific details on the implementation used in our study are available in [29]. In Chapters 4–7 we created a ranking that enabled us to make comparisons to features used in prior research.

3.4.3 **Scenarios and Feature Transformation**

Datasets used to detect malicious websites commonly contain class imbalances (i.e., the size of the malicious dataset and the size of the benign dataset often differ from one another). This is true for prior research and was true for our research as well. We acknowledge that our Dataset 1 was unbalanced and sought to examine whether and how

this affected the performance of our detection models. We used three sampling scenarios: 1) no-sampling (using the dataset as is); 2) over-sampling (incrementally over-sampling the malicious dataset to make the number of benign and malicious websites equal); and 3) under-sampling (sampling the benign dataset to lower the number of benign websites to equal the number of malicious websites). We performed over-sampling using the Synthetic Minority Over-Sampling Technique (SMOTE) [186] from [187], while under-sampling was random. We applied each of our machine learning algorithms to the three sampling scenarios, yielding several models for analysis (multiple models per sampling scenario). Class-balancing also was explored by changing the `class weight` [29] parameter of the models. However, this was shown to have little effect on performance and involved performing an exhaustive grid-search on the `weight` parameters.

In addition to the no-sampling, over-sampling, and under-sampling scenarios, we created two more scenarios using feature transformation techniques – feature transformation with feature selection (FT w/FS) and feature transformation with PCA (FT w/PCA). Feature transformation enabled us to create additional features using pair-wise arithmetic operations (addition, multiplication, and division). After creating these new features, we independently performed additional feature selection and PCA to identify relevant features and components, respectively. We performed feature transformation with pair-wise feature transformations (addition, multiplication, and division) with the help of a `Python` library, `featuretools` [188]. The additional feature selection included the use of correlation [189], SelectKBest (scoring function chi-square), recursive feature elimination (RFE), and SelectFromModel [29] to select a subset of features. We input the transformed features into these four techniques and

selected features that were identified by at least three of these four techniques. PCA created new features, also known as components, by reducing the features to "n" principal components that captured a large portion of variance in the data. We used two techniques to accomplish feature transformation – feature transformation with feature selection and feature transformation with PCA – applying them exclusively to the no-sampling scenario. With the addition of these two-feature transformation cases, we had several models for analysis (models repeated over the five scenarios, which consisted of three sampling scenarios and two feature transformation scenarios). Figure 3-3 below shows the scenarios we used in this study.

	No-sampling	Over-sampling	Under-sampling
Feature Selection	**X**	**X**	**X**
Feature Transformation with Feature Selection	**X**		
Feature Transformation with PCA	**X**		

Fig. 3-3. Several sampling and feature transformation scenarios were used throughout this research

In Chapters 4 and 6, we rebuilt these sets of models on two sets of features – those identified by our approach and those used in prior research. Doing so allowed us to compare the effect of the newly identified features in this research. Chapter 5 is unique from Chapters 4 and 6 in its focus on a set of features (URL structure) that have been extensively studied. Thus, we did not make a comparison between new features and features from prior research since prior research has covered many of the possible URL features and the distinction between those that are new and those that are from prior research is not clear.

3.5 Step 4: Tune and Cross-Validate

3.5.1 Hyperparameter Tuning and Cross-Validation

Once we had the performance metrics for each of the models in the respective scenarios, we performed cross-validation and hyperparameter tuning for two reasons: first, to improve the performance of the models, and second, to demonstrate the consistency of the models with and without hyperparameter tuning and cross-validation. We used a decision tree classifier as the base estimator and StratifiedKFold [190] for 10-fold cross-validation. We used ACC, Prec, Rec, and F1 score as potential scoring metrics. We performed hyperparameter tuning and cross-validation on the best model in each of the five scenarios (no-sampling, under-sampling, over-sampling, feature transformation with feature selection, and feature transformation with PCA).

3.5.2 Validation with Another Data Split

In our study, we trained and tested our models using an 80:20 split of train to test data. We used 80% of the dataset to train and used the remaining 20% to test our models. This approach is common in prior research. We used the same 20% to evaluate our models to ensure consistency. To further demonstrate that our results were not a product of our initial 80:20 split of training to testing data, we rebuilt our models in the various scenarios and performed hyperparameter tuning and cross-validation, starting with a 70:30 split of train to test data. We then compared these results to the tuned and cross-validated models built with the 80:20 split.

3.6 Step 5: Combine Features for Improved Detection

3.6.1 Combined Features in this Study

In this step, we built detection models over various scenarios on a set of features derived from all categories of the features in our study. We also built models with features exclusively from prior research.

3.6.2 Additional Detection Models

We built models using nine different supervised learning models and two models from unsupervised learning techniques on two feature sets – those identified in this dissertation and those from prior research. We performed feature selection by following the same procedure detailed in Section 3.3.2 of this chapter. The supervised learning models included KNN, AB, ET, RF, GB, XGB, BC, NNs, and a voting classifier (V) [191] built form the RF, ET, and GB. We excluded LR in this study because it consistently proved to be one of the worst performing models from the prior steps. We gathered feature importance from the AB, ET, RF, GB, and XGB algorithms. The unsupervised models included one-class SVMs and autoencoders [192], both of which have been used for malicious website detection.

3.6.3 Hyperparameter Tuning and Cross-Validation

We performed hyperparameter tuning and cross-validation as outlined in Step 5. However, we also varied the Scikit-Learn [29] `class weight` parameter, which penalized missed classifications for the positive (malicious) or negative (benign) classes in the classification. We varied the Scikit-Learn `class weight` parameter in this step as an alternative to sampling.

3.7 Step 6: Evaluate on Another Dataset

3.7.1 Model Application to a New Dataset (Dataset 2)

We then applied to a new dataset the best performing RF classifier built thus far (the classifier that performed well in our studies thus far and performed well in prior research). The new dataset (Dataset 2) consisted of the Alexa Top 1M domains. We defined malicious websites as those websites that were in the Alexa Top 1M and that also were identified in threat intelligence information provided by Cymon.io [193]. We defined benign websites as those from the Alexa Top 1M that were not present in the Cymon.io dataset. We directly applied the model trained from Step 5, captured the performance metrics, and explored any differences.

3.7.2 Retrain with Features Identified in Prior Studies (Section 3.3)

We also explored the capabilities of the features identified in our first dataset (Dataset 1) to another dataset (Dataset 2) by re-training a model based with the features identified from Dataset 1 on Dataset 2 and evaluating the detection ability of the new model on Dataset 2. We then evaluated the performance and determined whether new features derived from the newer dataset (Dataset 2) could be incorporated to improve detection.

3.7.3 Leverage Two Datasets for Training and Evaluation

In Step 6, we explored the use of two different datasets for training and evaluation. Furthermore, we made observations on the impact and feasibility of doing so.

3.8 Step 7: Explore Detection Performance Over Time

3.8.1 Measure the Performance of a Model Trained on Dataset 1 and Evaluated on Dataset 3

We required a well-performing model for evaluating detection performance over time. To that end, we first examined the performance of an RF model built on Dataset 1 and evaluated on Dataset 3. We evaluated how consistently the entries in the dataset were classified and how well the model performed.

3.8.2 Investigate the Impact of Model Retraining on Performance

We then investigated the impact of model re-training by re-training an RF model on the first snapshot of Dataset 3 and evaluating on the proceeding snapshots in Dataset 3. The model was trained using three sets of features – the identified features from Dataset 1 (the features in Chapter 7), the features used in prior research, and a new set of features re-selected on Dataset 3. We then re-trained on each week and evaluated the performance on subsequent weeks. Finally, we re-trained the model using all past data (instead of a single snapshot) and evaluated the model on the subsequent weeks.

3.8.3 Evaluate Website Change Over Time

The internet is a fast-changing environment and websites change over time. These changes can occur in areas that may influence detection models, including the features that are used for detection. Hence, we measured the change in the websites over time (based on the features used in our detection models). We used four tests –the t-test for related samples, the two-sample Kolmogorov-Smirnov (KS) test, the k-sample Anderson-Darling test [198]-[199], and the Kruskal Wallis H test [200]-[201] – summarized as follows:

- The related (dependent) t-test [194]-[195] tests whether means of two related samples are the same and the t-statistic is given by:

$$t = \frac{m}{s/\sqrt{n}}$$

where:

- `m` is the mean differences of all the paired measurements,
- `n` is the sample size, and
- `s` is the standard deviation of the differences

If the t-statistic is greater than a critical value, the null hypothesis of equal means can be rejected.

- The two-sample KS test [196]-[197] tests that two samples come from the same distribution. The KS statistic (*D* below) is expressed by:

$$D = |E_1(i) - E_2(i)|$$

where E_1 and E_2 are the empirical distributions for the two samples. We can reject the null hypothesis that the two samples come from a common distribution if the following is true:

$$D_{n,m} > c(\alpha)\sqrt{\frac{n+m}{n \cdot m}}$$

α	0.1	0.05	0.025	0.01	0.005	0.001
c(α)	1.22	1.36	1.48	1.63	1.73	1.95

where:

- D is the KS statistic,
- α is the significance level,
- c(α) is the critical value per significance level, and

- n and m are the sizes of the samples.

- The k-sample Anderson-Darling test [198]-[199] tests the null hypothesis that the populations for which two or more groups were drawn are identical. The Anderson-Darling (A) statistics is described below:

$$A = \frac{n-1}{n^2\,(k-1)} \sum_{i=1}^{k} [\frac{1}{n_i} \sum_{j=1}^{L} h_j \frac{\left(nF_{ij} - n_i H_j\right)^2}{H_j\left(n - H_j\right) - \frac{nh_j}{4}}]$$

where:

- A_i are the populations we are considering,
- n_i = the total number of data points from A_i,
- x_{ij} is the jth observation in ith group,
- n = the total number of data points for all n_i,
- L = the number of distinct data points in the combined sample,
- $z_* = z_1, z_2, \ldots z_L$ are the distinct values in the combined data set ordered from smallest to largest,
- h_j = number of values in the combined samples equal to z_j,
- H_j = number of values in the combined samples less than z_j plus one half the number of values in the combined samples equal to z_j,
- F_{ij} = number of values in the ith group (A_i) which are less than z_j plus one half the number of values in this group which are equal to z_j , and
- k = number of groups.

The null hypothesis is that the samples were drawn from the same population and can be rejected if the test statistic is greater than a critical value.

- The Kruskal Wallis H test [200]-[201] determines whether medians of two or more groups are different. The H statistic is given by:

$$H = \left[\frac{12}{n\,(n+1)} \sum_{j=1}^{c} \frac{T_j^2}{n_j}\right] - 3(n+1)$$

where:

- n = sum of sample sizes for all samples,
- c = number of samples,
- T_j = sum of ranks in the j^{th} sample, and
- n_j = size of the j^{th} sample.

If the H statistic is greater than a critical value, we can reject the null hypothesis that the medians are the same.

All four of these tests allowed us to determine whether two samples or sets of data came from a similar distribution and formed the basis for how we determined whether websites (and their features) have changed. Their application is discussed in detail in Chapter 9.

3.9 Summary

In this section we discussed our methodology. We started with a high-level description of our approach then discussed the steps taken in this dissertation. The first step included selection of three datasets used in the proceeding chapters. We then discussed our approach to discover features for malicious website detection through extensive feature consideration and through a process of feature selection. The next step in our methodology is the creation and evaluation of detection models from distinct types of features – webpage content, URL, and HTTP headers, with various learning algorithms in different scenarios. We further investigated detection performance by

performing tuning and cross-validation of the models. After this investigation, we performed studies to measure the detection performance when leveraging all three types of features (webpage content, URL, and HTTP headers) in this dissertation. We shifted emphasis in the later portion of this dissertation and performed an investigation of the effectiveness of the models built thus far and the features identified when applied to another dataset. We concluded our methodology with steps for our temporal study of malicious website detection.

Chapter 4: Webpage Content Features Analysis

4.1 Introduction

In this chapter, we explore an approach using only webpage content for three reasons. First, prior research places little emphasis on finding new features derived from webpage content to detect malicious websites, which can lead to potential missed detection opportunities. For example, the `<iframe>` HTML element has been considered a means of detecting malicious websites for more than 11 years without re-evaluation. Second, security operations centers (SOCs) or incident response teams can gather webpage content features with little effort and incorporate them into signatures to detect malicious websites. Third, most prior research focused on detecting either phishing websites or drive-by downloads. While these results were promising, they required *a priori* knowledge of the target website, which is not a viable solution for a SOC or an incident response team. We then evaluated the ability of webpage content features in order to detect malicious websites on a diverse dataset (Dataset 1) containing several types of malicious websites to gain insight into their performance when *a priori* knowledge is not available. Our contributions are outlined below.

- We re-evaluated the importance of features for detection of malicious website from prior research and provided a ranking of webpage content features to detect malicious websites.
- We created an approach using webpage content to identify 26 features, 17 of which were introduced in our study, to detect malicious websites with an average ACC, AUC, and MCC of 89.15%, 0.867, and 0.641, respectively, across all sampling and feature transformation scenarios.

- Our approach identified 26 features, with 17 of them introduced in our study, whose models produced an average MCC that was 0.005 higher than models built with features identified in prior research and did so with 48% fewer features.
- We identified features, both new and from prior research, that showed promise for detecting websites involved in phishing attacks, drive-by downloads, and C2 activities.

4.2 Related Research

Researchers have used features gathered from webpage content – both the HTML and the JavaScript on a webpage – to detect malicious websites separately and collectively. Provos et al. [59] examined drive-by downloads, commonly enabled by the `<iframe>` HTML element. Zhang et al. [24] looked for the `<input>` tag accompanied by the words "credit card" and "password" as indicators of phishing websites. Xiang et al. [46] built a framework to detect phishing websites using features gathered from the URL structure and HTML on the webpage. Both [24] and [46] used approaches for phishing website detection that relied on the assumption that phishing websites will often try to "trick" a user into entering sensitive information. Whittaker et al. [42] applied statistics to use of the password field and to links on the webpage to build a classifier with a TPR of 95% against websites involved in phishing attacks. Marchal et al. [91] used the links on the webpage, in conjunction with URL features and the Alexa ranking of the domain, as a set of features to detect phishing websites and achieved an AUC of 0.999 for English webpages. Arab and Sohrabi [202] used a list of website features

derived from many aspects of a website to create clusters for phishing website detection and achieved 99% accuracy on their dataset of 200 websites.

Other authors focused solely on gathering features from the JavaScript on the webpage. Curtsinger et al. [68] detected JavaScript malware with an AST based approach by instrumenting the browser with a "de-obfuscator" to get a better representation of the actual JavaScript on the webpage and produced an FPR of 0.0003%. JStill [69] used the fact that malicious JavaScript is often obfuscated and used practical examples on malicious JavaScript techniques, including data obfuscation, ASCII encoding, and logical structure obfuscation and produced an FPR of 1.75% and 0.53%. Researchers [119] used JaSt to detect and analyze obfuscated JavaScript, using entirely static analysis that yielded an ACC of nearly 99.5% when used with an RF classifier.

HTML and JavaScript have often been studied independently and have also been combined for malicious website detection. In an influential paper, [23] gathered features from the `<script>` and `<frame>` elements to achieve an FPR of 5.88% and an FNR of 46.15%. Researchers [45], with Prophiler, extracted both HTML and JavaScript features to create a "fast filter" for detecting drive-by downloads and achieved an FPR of 9.88% and FNR of 0.77%. Researchers [47] and [165] collected suspicious HTML features along with the counts of suspicious JavaScript methods such as `eval()`, `charCodeAt()`, `unescape()`, and others that are known to be associated with malicious JavaScript. They achieved accuracies of 97.8% and 96.5%, respectively. Authors [49] and [66] used the respective counts of suspicious JavaScript methods and specific HTML tags in their feature collection to achieve an ACC of 96.39% and an AUC of 0.891, respectively.

4.3 Research Questions 1–4

We created four research questions to explore the effectiveness of this approach and the webpage content features we identified as features for malicious website detection. These questions focused on using webpage content features – that is, the HTML and JavaScript on the webpage – as the sole source of features for detection of whether the website was malicious.

4.3.1 Research Question 1

Our first question aided in determining how well our approach aligned with or differed from prior research. Some previous researchers used webpage content to detect malicious websites, but did not evaluate features that have not demonstrated potential for malicious website detection. We considered 17,746 features in total, gathered from the HTML and JavaScript on the webpage. While no definitive list of webpage content features currently exists, certain HTML and JavaScript features have been commonly reused in prior research. We hypothesized that our approach, which considered 17,746 features, many of which had never been studied for malicious website detection, would identify new features that were important to the detection of malicious websites. Research Question 1 is stated as follows:

RQ1: How do the features identified compare with prior research?

4.3.2 Research Question 2

Our second research question investigated whether the incorporation of these features improved malicious website detection. This was done by comparing the MCCs for models built with the features identified by our approach to the MCCs of models built with features from prior research. We added assurance to our approach by performing

feature transformation techniques with feature selection and PCA, comparing the respective MCCs. Hence, RQ2 is stated as follows:

RQ2: Do the additional features identified improve malicious website detection?

4.3.3 Research Question 3

Our third research question focused on the robustness of our approach by investigating how our results changed in different sampling scenarios – that is, whether our approach yielded consistent results in the cases of no-sampling, over-sampling, and under-sampling of our dataset. In security research, class imbalances between the benign and malicious datasets are common. We also had an imbalance of malicious and benign websites in our dataset. Hence, we state RQ3 as follows:

RQ3: Do our results change with no-sampling, under-sampling, and over-sampling scenarios?

4.3.4 Research Question 4

Our fourth research question enabled us to explore additional validation of our results by performing hyperparameter tuning and cross-validation in an attempt to improve our results. Hyperparameter tuning and cross-validation could enable us to build better detection models. RQ4 is stated as follows:

RQ4: Does hyperparameter tuning and cross-validation improve our results?

4.4 Feature Consideration

4.4.1 JavaScript Methods

From our literature review, we observed that the presence and counts of JavaScript methods are often used as a JavaScript feature for malicious website detection. Method counts are defined as the number of invocations of a specific method found on a

webpage. For example, extracting the method count for the method `eval` on the following code snippet would result in a value of 2 – that is, we count two invocations of the method `eval`:

```
console.log(eval('3 + 2') === eval('5'));
```

JavaScript methods of interest from previous research fall into three loose categories – 1) obfuscation methods, 2) suspicious methods, and 3) methods that act on the Window or DOM objects. These categories are considered loose because potential exists for a method to be found in more than one category. For example, obfuscation methods are often considered suspicious, but suspicious methods exist that are not related to obfuscation. In addition, methods that act on the DOM and Window objects can also be considered suspicious, yet they maintain some uniqueness because they act upon the DOM and Window objects.

4.4.1.1 Obfuscation Methods

Obfuscation is a technique used by malicious JavaScript writers to hinder analysis of their code, thus making it more difficult to analyze it and to detect it as malicious JavaScript code. Obfuscated JavaScript is challenging to read, but it contains certain characteristics useful for determining whether it is malicious. These obfuscation characteristics include use of specific methods such as `replace` and `unescape`. The snippet of code below from [203] shows normal JavaScript and its obfuscated equivalent.

- No obfuscation:

```
alert( 'Hello, world!' );
```

- Obfuscation :

```
var _0x1dc7 = ["\x48\x65\x6C\x6C\x6F\x2C\x20\x77\x6F\x72\x6C\x64\x21"];alert(_0x1dc7[0])
```

4.4.1.2 Suspicious Methods

Methods are considered suspicious for many reasons, including their presence in specific types of attacks. The code snippet below from [204] uses events to send a user to a fake website when they try to go to the previous webpage.

```
function addBackClickAd(options) {a
  if (options['backClickAd'] && options['backClickZone'] && typeof
window['history']['pushState'] === 'function') {
    if (options['backClickNoHistoryOnly'] && window['history'].length >
1) {
      return false;
    }
    // pushes a fake history state with the current doc title
    window['history']['pushState']({exp: Math['random']()},
document['title'], null);
    var createdAnchor = document['createElement']('a');
    createdAnchor['href'] = options['url'];
    var newURL = 'http://' + createdAnchor['host'] + '/afu.php?zoneid=' +
options['backClickZone'] + '&var=' + options['zoneId'];
    setTimeout(function () {
      window['addEventListener']('popstate', function (W) {
        window['location']['replace'](newURL);
      });
    }, 0);
  }
}
```

4.4.1.3 Methods that Act on the Window or DOM Objects

The DOM is the internal representation of the webpage document and the Window object represents the browser window. It is common for malicious JavaScript to manipulate or misuse properties of both the DOM and Window objects to facilitate attacks. The example below shows malicious JavaScript that manipulates the DOM from [205].

```
(function () {
    var qk = document.createElement('iframe');  // creating an
iframe

    qk.src = 'http://xxx.tld/wp-includes/dtd.php';  // pointing
it at a webpage

    /*
    making the iframe only take up a 1px by 1px square
    in the top left-hand corner of the web page it is injected
into
    */
```

```
    qk.style.position = 'absolute';
    qk.style.border = '0';
    qk.style.height = '1px';
    qk.style.width = '1px';
    qk.style.left = '1px';
    qk.style.top = '1px';

    /*
    Adding the iframe to the DOM by creating a <div> with an ID
of "qt"
    (If the div has not been created already)
    */
    if (!document.getElementById('qk')) {
        document.write('<div id=\'qk\'></div>');
        document.getElementById('qk').appendChild(qk);
    }
})();
```

Table 4-1 lists commonly studied JavaScript methods involved in obfuscation, considered suspicious, and that act upon the DOM and the Window objects.

Table 4-1.
Certain JavaScript Methods Were Considered Suspicious
and Have Been Studied in Prior Research

Commonly Studied JavaScript Methods	
Method	***Motivation***
`createElement`	This method modifies the data object model (DOM).
`write`	This method modifies the DOM, writes a string.
`charCodeAt`	This method is considered suspicious and has been used in JavaScript obfuscation.
`Concat`	This method manipulates strings and is associated with obfuscation.
`escape`	This method is considered suspicious and has been used in obfuscation.
`eval`	This method is considered suspicious and enables the execution of a string as code.
`exec`	This method is considered suspicious and can be used in obfuscation.
`fromCharCode`	This method has been associated with obfuscation.
`link`	This global method is considered suspicious and has appeared in many types of attacks.
`parseInt`	This method has been associated with malicious combinations of methods.
`replace`	This method is commonly used in obfuscation. This method has also been shown to be used in conjunction with shellcode.
`search`	This global method is considered suspicious and has appeared in many types of attacks.
`subString`	This method associated with string manipulation and obfuscation.
`unescape`	This method is considered suspicious and has been used in obfuscation. This method has also been shown to be used in conjunction with shellcode.
`addEventListener`	Event attachments can be considered suspicious under certain circumstances.
`setInterval`	The method is involved in executing code after a certain time interval and has been used in drive-by download attacks.
`setTimeout`	The method is involved in executing code after a certain time interval and has been used in drive-by download attacks.

Although we extracted the counts for the methods in Table 4-1, we also included another 384 methods found on Mozilla Developer Network (MDN) [206] and W3Schools [207]. MDN and W3 were consulted because they are intended for JavaScript developers and contain extensive and up-to-date information on JavaScript. The additional 384 methods were chosen for our study because they are related to previously studied methods, albeit there is little published research about the use of these methods for detecting malicious websites. For example, only two methods that act on the DOM have been studied in previous research reviewed; we added an additional 46 DOM methods to

our feature set in addition to the two DOM methods, `write` and `createElement`, listed in Table 4-1. With our approach, we captured methods that are relevant as well as methods from previous research, and we explored other methods that may be relevant for detecting malicious websites. For a complete list of all JavaScript method counts collected in this chapter, please see Appendix B.

4.4.2 **HTML Characteristics**

Another feature-rich aspect of webpage content is the HTML. When a browser loads a webpage, it uses the HTML to determine how to represent the webpage to the user. HTML defines the structure of the webpage, including visual characteristics, specific elements, and attributes. It consists of elements – also referred to as tags – specified by `<element_name>` and of attributes specified within an element. We refer to an attribute within a specific element as an element-attribute pair.

The HTML code example below represents a webpage that specifies links to two websites – CNN and Google.

```
<html>
      <body>
             <a href=http://www.cnn.com/>
             <br/><br/>
             <a href=http://www.google.com/>
       </body>
</html>
```

Although this example is small, it contains several features we can collect: the count of `<body>`, `<a>`, and `<br>` elements, as well as details about the `href` attribute in the `<a>` element (also referred to as the `a_href` element-attribute pair). Running this HTML through the feature collector we developed created a feature vector like the one shown below.

```
<a> count=2
<body> count=1
```

```
<br> count=1
Total out of domain URLs=2
Total HTML Tags=5
Total href attributes=2
<a href =”http*”>=2
<a href =”*.com”>=2
```

This vector can be interpreted as: “This HTML contains five tags, two `href` attributes, one `<body>` element, two `<a>` elements, and one `<br>` element. Two of the links on the webpage point to resources outside of the domain, two of the `href` attributes on the `<a>` element point to a resource specified over the HTTP protocol, and two `href` attributes point to a `.com` URL.” Other research typically counts specific elements such as `<iframe>`. We took a more expansive approach, expanding our collection of element counts to include many HTML elements. Please see Appendix B for a complete listing of elements collected in this chapter. Additionally, we expanded analysis of element-attribute pairs that specify resources via URLs. URLs specified on a webpage are interesting because they can reference a resource and have many properties that translate to potential features. These properties can be extracted and used for malicious website detection. While we included element-attribute pairs from previous research, we also expanded and analyzed webpage URLs and additional element-attribute pairs not previously studied. For a complete listing of element-attribute pairs we collected in this chapter, please refer to Appendix C. Table C-1 in Appendix C also specifies the attributes for additional URL analysis for the respective elements. This is specified in the last column of Table C-1. The last feature we collected is the number of small elements of a specified HTML type. Previous research captured the presence of small `<iframe>`s and `<frame>`s. We did the same, but we also included other elements that have size

attributes. An element is considered small if it has a height or width of less than or equal to two pixels.

4.5 Feature Collection

We wrote our collection scripts in `Python` and used `Pyselenium` [208] to fetch the webpage and retrieve the information. `Pyselenium` was chosen for its ability to parse the HTML and extract values and attributes. We extracted JavaScript method counts by searching for a method call on the webpage to speed up extraction for potential implementation into a detector. HTML feature extraction was more complex because a page can have several instances of a specific element and those elements can contain various attributes. Furthermore, not all attributes are guaranteed to be present in each element. To account for this, we created a simple algorithm to aide our HTML feature extraction. The pseudo-code for HTML feature extraction is shown below. Special elements are specific elements where we extract additional attributes such as features regarding the resource URLs (for example `href`), whether the element is "small," etc.

```
elements = ALL_HTML_ELEMENTS
for elem in elements:
 count = get_total_element_count(elem)
 if elem is special_element:
    special_attributes = get_special_attributes(elem)
      for special_attribute in special_attributes:
        extract_attribute(elem, special_attribute)
```

4.6 Learning, Feature Selection, and Sampling Techniques in Webpage Content Analysis

4.6.1 Feature Elimination Process

We then sought to shrink our feature set of 17,746 webpage content features to a smaller, more useful set of no more than 50 features. The number 50 was chosen subjectively, but it also is approximately the number of features used to detect malicious

websites in prior research. For example, [45] used 77 features with Prophiler, while [47] used 30 features with BINSPECT. We followed the approach outlined in Section 3.3.2, determining which features had strong association with the dependent variable, whether the website was malicious, and which had no relationship or a weak relationship with the dependent variable. We identified and removed features specific to our dataset, as well as features that are the same for most of the dataset. Hence, we removed features that had the same value 95% or more of the time. This eliminated 17,525 features and left us with 221 features. We then evaluated the remaining 221 features to identify those that had a high multicollinearity. Removing features with high multicollinearity was required in order to ensure that we analyzed a set of independent features. We quantified collinearity with the VIF [177]. First, we computed the VIF for each feature. We then identified features that had a VIF > 5, as used in [179]. Among our list of features with a VIF greater than five, we then determined which features had similar VIF values, thereby showing that they had correlations similar to those of the other variables and had high correlation to each other. We considered a high correlation to be a correlation of greater than 0.7, as used in [180]. Among the highly correlated features with similar VIF values, we dropped the feature with the higher VIF. This process resulted in 43 features removed, leaving us with 178 features.

Since we had more than 50 features remaining, we continued to remove features using the XGB algorithm. XGB is a GB algorithm that also computes feature importance. To remove additional features, we first calculated the feature importance for each feature in the set of 178. This was done by building a model from a 70:30 split of training to test data. Once we had the importance values for the 178 features, we then iteratively input

each feature importance as a threshold to the SelectFromModel technique [29], a transformer used to select features based on their weights to produce a set of features. This produced a set of features for each threshold. We then used each set of features associated with each threshold and rebuilt our XGB models to obtain an ACC for each set of features. At this point in our analysis, we have a list of sets consisting of a threshold "thresh," number of features "n," set of features "f," and an ACC. An example is below.

```
                    …
Thresh=0.009, n=31, f=[..], Accuracy: 90.58%
Thresh=0.010, n=26, f=[..], Accuracy: 90.62%
Thresh=0.010, n=26, f=[..], Accuracy: 90.62%
Thresh=0.010, n=26, f=[..], Accuracy: 90.62%
Thresh=0.013, n=23, f=[..], Accuracy: 90.58%
                    …
```

There are three entries for a threshold of 0.010 because the threshold 0.010 appeared three times in the list of feature importance values for the 178 features. We then iterated through the list of sets with "n" decreasing and identified relative maxima in the respective ACC. We found a relative maximum at $n = 26$ and used the features associated with this relative maximum as our final feature set.

4.6.2 Machine Learning Models, Sampling, and Feature Transformation

To ensure that we identified a relevant set of features, we evaluated the effectiveness for detecting malicious websites by building eight models using supervised machine learning algorithms discussed in Section 3.4.1.

For all models, we split training and testing data using an 80:20 ratio, a common train/test split for data. Our overall dataset is imbalanced, with 34,778 benign websites and 5,931 malicious websites. To address this and to ensure that our results were not the product of our benign-to-malicious website ratio, we performed the sampling procedure outlined in Section 3.4.3.

For no-sampling, we used 27,822 benign websites and 4,745 malicious websites in our training set. Under-sampling resulted in 4,745 malicious websites and 4,745 benign websites in the under-sampled training set. For over-sampling, we arrived at a balanced training set with 27,822 benign websites and 27,822 malicious websites.

The websites used in the testing dataset remained consistent across all machine learning models and sampling approaches for the training data so that we could compare model results and identify whether any single sampling technique led to a better model. We ensured that there was no overlap between any training and testing data. We also built models in feature transformation scenarios as discussed in Section 3.4.3. Figure 3-3 from Chapter 3 provides a summary of the feature selection and sampling techniques.

4.7 Results

4.7.1 RQ1: How do the Features Identified Compare with Prior Research?

RQ1 compared the features identified in our approach with those from prior research in terms of ability to detect malicious websites. To examine this question, we leveraged our four ensemble methods (RF, AB, ET, and BC), all of which captured the notion of feature importance. The higher the importance, the more the feature contributed toward determining whether the website was malicious. The identified 26 features are shown below in Table 4-2, along with their rank and importance, separated by a ":" in the no-sampling, over-sampling, and under-sampling scenarios. Shaded rows designate new features we identified in our research. Unshaded rows designate features studied in prior research for identifying malicious websites or traffic.

Table 4-2.
Feature Selection Identified 26 Webpage Content Features for Detection

26 Identified Webpage Content Features Ranked			
Feature	***No-sampling***	***Over-sampling***	***Under-sampling***
Total HTML Tags	1: 0.3206	1: 0.2705	1: 0.2239
Total href attributes	2: 0.1025	2: 0.1190	2: 0.1723
<link href> OoD	3: 0.0644	3: 0.0943	3: 0.1018
<p> count	4: 0.0567	5: 0.0601	4: 0.0642
<a href="https*">	5: 0.0554	8: 0.0403	6: 0.0581
Count of <meta> tag	6: 0.0515	6: 0.0471	8: 0.0340
<script_async=true>	7: 0.0462	7: 0.045	5: 0.0634
<link type="text/css">	8: 0.0298	9: 0.0327	11: 0.0257
<script src> OoD	9: 0.0289	14: 0.0141	7: 0.0535
<link href="http*">	10: 0.0271	11: 0.0224	10: 0.0283
push()	11: 0.0258	4: 0.0627	9: 0.0325
<link href="*.css">	12: 0.0258	12: 0.0205	13: 0.0125
indexOf()	13: 0.0175	25: 0.0071	16: 0.0119
<form action="http*>	14: 0.0168	19: 0.012	12: 0.0136
count	15: 0.0151	15: 0.0132	18: 0.0114
<iframe src="https*">	16: 0.015	10: 0.0271	24: 0.0078
Count of tag	17: 0.0141	16: 0.0131	19: 0.0093
setTimeout()	18: 0.0136	26: 0.0066	15: 0.0121
<a href="*.com">	19: 0.0133	13: 0.0186	20: 0.0090
document.write()	20: 0.0112	17: 0.0129	22: 0.0084
addEventListener()	21: 0.0096	20: 0.011	14: 0.0124
get()	22: 0.0093	21: 0.0107	26: 0.0023
<link type="application/rsd+xml">	23: 0.0079	22: 0.0103	21: 0.0088
find()	24: 0.0077	24: 0.0078	25: 0.0035
<link rel="shortlink">	25: 0.0073	23: 0.0085	23: 0.0080
replace()	26: 0.0069	18: 0.0123	17: 0.0114

We repeated this exercise on features from prior research, with their respective ranking and importance shown in Table 4-3 below.

Table 4-3.
50 Webpage Content Features from Prior Research Showed
Inconsistent Rank in Sampling Scenarios

50 Webpage Content Features from Prior Research Ranked			
Feature	***No-sampling***	***Over-sampling***	***Under-sampling***
Total HTML Tags	1: 0.3190	1: 0.2694	1: 0.2452
Count of <meta> tag	2: 0.0620	7: 0.0437	7: 0.0507
<a href> OoD	3: 0.0583	2: 0.0688	2: 0.1071
Total href attributes	4: 0.0534	6: 0.0509	5: 0.0665
Count of tag	5: 0.0462	17: 0.0182	11: 0.0292
Count of <a> tag	6: 0.0457	4: 0.0669	3: 0.0734
<link href> OoD	7: 0.0437	3: 0.0671	4: 0.0684
<script src> OoD	8: 0.0408	5: 0.0563	6: 0.0564
Count of <link> tag	9: 0.0330	13: 0.0234	10: 0.0296
Total <img src>	10: 0.0307	9: 0.0342	8: 0.0415
<img src> OoD	11: 0.0252	11: 0.0285	13: 0.0218
Count of <title> tag	12: 0.0245	15: 0.0196	18: 0.0086
createElement()	13: 0.0238	8: 0.0395	12: 0.0276
Count of <img> tag	14: 0.0207	12: 0.0255	9: 0.0351
<script type= ”text/javascript”>	15: 0.0198	16: 0.0191	14: 0.0182
Count of <input> tag	16: 0.0164	20: 0.0094	20: 0.0069
Count of <iframe> tag	17: 0.0152	14: 0.0223	15: 0.0166
<form action> OoD	18: 0.0150	10: 0.0341	16: 0.0119
replace()	19: 0.0136	19: 0.0134	17: 0.0110
Count of <style> tag	20: 0.0077	26: 0.0058	21: 0.0064
escape()	21: 0.0075	22: 0.0079	22: 0.0063
addEventListener()	22: 0.0072	21: 0.0086	19: 0.0077
setTimeout()	23: 0.0072	24: 0.0064	24: 0.0061
substring()	24: 0.0071	33: 0.0020	33: 0.0018
concat()	25: 0.0064	23: 0.0065	34: 0.0013
document.write()	26: 0.0063	31: 0.0031	32: 0.0025
fromCharCode()	27: 0.0059	27: 0.0056	25: 0.0059
<img srcset> OoD	28: 0.0058	25: 0.0063	23: 0.0062
search()	29: 0.0054	28: 0.0052	26: 0.0054
charCodeAt()	30: 0.0053	39: 0.0003	43: 0.0003
<audio src> OoD	31: 0.0051	29: 0.0052	27: 0.0052
<iframe src> OoD	32: 0.0033	32: 0.0025	28: 0.0043
parseInt()	33: 0.0030	18: 0.0136	31: 0.0030
<base href> OoD	34: 0.0021	30: 0.0042	30: 0.0035
unescape()	35: 0.0021	34: 0.0020	29: 0.0037
eval()	36: 0.0010	35: 0.0009	38: 0.0005
Count of <frame> tag	37: 0.0008	43: 0.0003	39 0.0005
exec()	38: 0.0007	38: 0.0005	35: 0.0007
Count of <object> tag	39: 0.0006	37: 0.0006	37: 0.0005
<frame src> OoD	40: 0.0006	42: 0.0003	41: 0.0004
<embed src> OoD	41: 0.0005	40: 0.0003	40: 0.0004
hidden <iframe>	42: 0.0004	36: 0.0008	36: 0.0006
Count of <embed> tag	43: 0.0003	41: 0.0003	42: 0.0003
<area href> OoD	44: 0.0002	46: 0.0001	46: 0.0001
<object data> OoD	45: 0.0002	44: 0.0002	45: 0.0002
setInterval()	46: 0.0001	45: 0.0001	47: 0.0001
link()	47: 0.0001	48: 0	44: 0.0003
<source src> OoD	48: 0.0001	47: 0.0001	48: 0.0001
<video src> OoD	49: 0	49: 0	50: 0
<source srcset> OoD	50: 0	50: 0	49: 0

4.7.1.1 Features Identified in Previous Research

Table 4-2 displays the 9 of 26 identified features that have been studied in previous research. They can be grouped and summarized in the manner outlined below.

- Two of the nine previously studied features centered around the number of HTML tags on the webpage. Tag counts were useful for identifying phishing websites in prior research. The `<meta>` tag was specifically used.
- Three of the nine previously studied features were gathered from the links and URLs found on the webpage. Six features are JavaScript methods studied in relation to malicious website detection. Links and URLs on the page were of particular interest if they pointed to out-of-domain (OoD) resources and were of interest since they could specify additional content that may be malicious without including malicious contents on the specific webpage.
- The final four features from prior work were counts of JavaScript methods that are considered suspicious or have been associated with JavaScript obfuscation. Another was a method that acts on the Window object,

4.7.1.2 New Features Identified

Table 4-2 identifies the 17 of 26 identified features that, to the best of our knowledge, have not been studied in prior research. They can be grouped and summarized in the manner outlined below.

- Three of the features were counts of tags that have not been studied for malicious website detection. Although tags have been studied, these three, to our knowledge, have not been selected for study in prior research.
- Four of the features were counts of additional JavaScript methods that are not common in studies to detect malicious websites.
- Six of the new features were gathered from the URLs specified in tags on the page.

- The remaining four features were specific values for attributes in several HTML tags.

4.7.1.3 Features Ranking Analysis

For the features identified by our approach in Table 4-2, the top three features are consistent and have the same rank in all scenarios. They are the count of all tags, the count of all `href` attributes, and the number of out-of-domain OoD `href` attributes in the `<link>` tag, having a total importance of 0.4875, 0.4838, and 0.4980, respectively. The features identified by our approach account for 40.61%, 41.97%, and 41.53%, respectively, of the total feature importance in the sampling scenarios in Table 4-2. Five of the features are counts of tags and account for 0.4016 average feature importance. Eight of the features are counts of JavaScript methods and account for 0.109 average feature importance. Seven of the features are related to the URLs on the webpage and account for 0.3577 average feature importance. The final six features are specific attribute values and account for 0.1313 of total feature importance.

When considering the 50 features from prior research in Table 4-3, in all three sampling scenarios, the total HTML tags (the first feature listed in Table 4-3) accounts for the most importance (0.3190, 0.2694, and 0.2452, respectively) and the importance difference between the first and second ranked feature is larger than the difference between the any other two consecutively ranked features. Thirteen of the 50 features are associated with tag counts, 17 are specific JavaScript method counts, and 16 are gathered from URLs on the webpage, two with specific values of attributes, and four with the counts of specific attributes. On average, we found that the most important features studied in prior research were gathered from counts of tags, URLs on the webpage,

counts of JavaScript methods on the webpage, and other specific attributes found in tags on the webpage accounting for an average total feature importance of 53%, 34%, 10%, and 2%, respectively.

Our approach identified 26 features, nine of which are from prior research, while the other 17, to the best of our knowledge, were new. The nine features account for roughly 40% of the total feature importance. For the 26 features identified by our approach, the top three are consistent across sampling scenarios and account for roughly half of the total feature importance.

4.7.2 RQ2: Do the Additional Features Identified Improve Malicious Website Detection?

We then investigated the performance of models built in our study in sampling and feature transformation scenarios. To do so, we built two sets of models with the 26 features identified by our approach and with the 50 features from prior research. We evaluated performance for the test dataset when using the no-sampling, under-sampling, and over-sampling training sets. Table 4-4 provides the FPR and FNR, the ACC, the AUC, and MCC for the 26 and 50 features and are separated by a "/." We focused on MCC to drive the discussion because MCC is a balanced metric that considers the four quadrants of the confusion matrix and works well even when the dataset is imbalanced. Table 4-5 provides the Prec and Rec of the respective models. In addition to sampling scenarios, we performed two sets of feature transformations on the 26 features identified by our approach and the 50 features gathered from prior research. These results are shown in Tables 4-4 and 4-5 below.

Table 4-4.
Identified Webpage Content Features Slightly Outperformed Features from Prior Research

Model Performance (50 Features from Prior Research / 26 Identified Features) in Sampling Scenarios															
	No-sampling					***Over-sampling***					***Under-sampling***				
Model	***FPR***	***FNR***	***ACC***	***AUC***	***MCC***	***FPR***	***FNR***	***ACC***	***AUC***	***MCC***	***FPR***	***FNR***	***ACC***	***AUC***	***MCC***
KNN	0.0844/ 0.0865	0.1239/ 0.1264	0.9097/ 0.9075	0.8958/ 0.8934	0.7003/ 0.6942	**0.0399**/ 0.0423	0.8344/ 0.8261	0.8427/ 0.8419	0.5628/ 0.5657	0.1899/ 0.1937	0.0409/ **0.0397**	0.8569/ 0.8377	0.8386/ 0.8424	0.5510/ 0.5612	0.1576/ 0.1860
LR	0.0861/ 0.0961	0.2262/ 0.1797	0.8931/ 0.8915	0.8437/ 0.8620	0.6246/ 0.6386	0.1814/ 0.1870	0.0532/ **0.0474**	0.8375/ 0.8335	0.8826/ 0.8827	0.5956/ 0.5925	0.1936/ 0.1943	0.0474/ 0.0482	0.8279/ 0.8271	0.8794/ 0.8786	0.5843/ 0.5828
RF	**0.0805**/ 0.0835	0.1198/ 0.1148	**0.9136**/ 0.9118	0.8998/ **0.9008**	**0.7110**/ 0.7083	0.0998/ 0.0902	0.0898/ 0.0890	0.9016/ **0.9099**	0.9051/ **0.9103**	0.6944/ **0.7131**	0.1201/ 0.1047	0.0715/ 0.0798	0.8870/ **0.8989**	0.9041/ 0.9076	0.6718/ **0.6925**
AB	0.0808/ 0.0847	0.2229/ 0.1747	0.8981/ 0.9019	0.8481/ 0.8702	0.6378/ 0.6642	0.1488/ 0.1337	0.0773/ 0.0673	0.8617/ 0.8760	0.8868/ 0.8994	0.6233/ 0.6530	0.1533/ 0.1504	0.0673/ 0.0557	0.8593/ 0.8635	0.8896/ 0.8969	0.6238/ 0.6360
GB	0.0806/ 0.0819	0.1647/ 0.1505	0.9069/ 0.9078	0.8772/ 0.8837	0.6794/ 0.6867	0.1234/ 0.1110	0.0790/ 0.0798	0.8830/ 0.8935	0.8987/ 0.9045	0.6612/ 0.6814	0.1328/ 0.1312	0.0673/ 0.0657	0.8768/ 0.8784	0.8998/ 0.9015	0.6543/ 0.6580
ET	0.0814/ 0.0842	0.1222/ 0.1156	0.9125/ 0.9110	0.8981/ 0.9000	0.7075/ 0.7063	0.1000/ 0.0904	0.0890/ 0.0956	0.9016/ 0.9087	0.9054/ 0.9069	0.6947/ 0.7079	0.1110/ 0.1118	0.0698/ 0.0806	0.8949/ 0.8927	**0.9095**/ 0.9037	0.6882/ 0.6795
BC	0.0817/ 0.0844	0.1206/ 0.1156	0.9125/ 0.9109	0.8988/ 0.8999	0.7081/ 0.7060	0.1028/ 0.0939	0.0840/ 0.0881	0.8999/ 0.9069	0.9065/ 0.9089	0.6930/ 0.7065	0.1161/ 0.1134	0.0748/ 0.0782	0.8899/ 0.8917	0.9044/ 0.9041	0.6762/ 0.6786
NN	0.0880/ 0.0985	0.1422/ **0.1039**	0.9039/ 0.9006	0.8848/ 0.8987	0.6804/ 0.6870	0.1342/ 0.1269	0.0565/ 0.0632	0.8771/ 0.8824	0.9045/ 0.9049	0.6595/ 0.6665	0.1693/ 0.1775	0.0457/ **0.0341**	0.8489/ 0.8436	0.8924/ 0.8941	0.6166/ 0.6138

Table 4-5.
Identified Webpage Content Features Slightly
Outperformed Features from Prior Research (cont.)

Model Performance (50 Features from Prior Research / 26 Identified Features) in Sampling Scenarios						
	No-sampling		***Over-sampling***		***Under-sampling***	
Model	***Prec***	***Rec***	***Prec***	***Rec***	***Prec***	***Rec***
KNN	0.6424/ 0.6359	0.8760/ 0.8735	0.4180/ 0.4155	0.1655/ 0.1738	0.3771/ 0.4140	0.1430/ 0.1622
LR	0.6086/ 0.5964	0.7737/ 0.8202	0.4747/ 0.4686	0.9467/ **0.9525**	0.4600/ 0.4588	0.9525/ 0.9517
RF	**0.6542**/ 0.6472	0.8801/ 0.8851	0.6121/ **0.6362**	0.9101/ 0.9109	0.5723/ **0.6033**	0.9284/ 0.9201
AB	0.6247/ 0.6278	0.7770/ 0.8252	0.5177/ 0.5470	0.9226/ 0.9326	0.5130/ 0.5208	0.9326/ 0.9442
GB	0.6419/ 0.6421	0.8352/ 0.8494	0.5636/ 0.5892	0.9209/ 0.9201	0.5487/ 0.5521	0.9326/ 0.9342
ET	0.6512/ 0.6450	0.8777/ 0.8843	0.6120/ 0.6338	0.9109/ 0.9043	0.5918/ 0.5874	0.9301/ 0.9193
BC	0.6508/ 0.6446	0.8793/ 0.8843	0.6066/ 0.6270	0.9159/ 0.9118	0.5797/ 0.5846	0.9251/ 0.9217
NN	0.6278/ 0.6115	0.8577/ **0.8960**	0.5488/ 0.5610	0.9434/ 0.9367	0.4939/ 0.4851	0.9542/ **0.9658**

Without sampling, the MCC was slightly higher for four of the eight models (LR, AdaBoost, GB, and NN), when considering the 26 features instead of the 50 previously studied features (on average, 0.6865 for the 26 features and 0.6812 for the 50 features). When over-sampling, the average MCC increased (0.6144 for 26 the features and 0.6015 for the 50 features) when considering 26 features instead of the previous studied 50 features. With over-sampling, the MCC was higher for all models except LR when considering the 26 features instead of the 50 previously studied features. When under-sampling, the average MCC increased (to 0.5910 for the 26 features and to 0.5842 for the 50 features). With under-sampling, the average MCC was higher for all eight models except LR, ET, and NN for the 26 features versus the 50 previously studied features. In each of our sampling scenarios, we observed overall improvement when building models with our 26 identified features compared to the 50 previously studied features. Although the improvement was not large, it was achieved with roughly half of the features, 17 of

which are not commonly used for malicious website detection. This suggests that additional features, outside of those identified in prior research, should be explored for their use in malicious website detection.

We also performed feature transformation with the process in Section 3.4.3 on the 26 features to investigate combinations of features that could improve performance and to evaluate the effects on the models. The 26 features were transformed into 1,326 feature combinations. We then performed feature selection on these feature combinations, using four different techniques: correlation, SelectKBest (scoring function chi-square), RFE, and SelectFromModel [29]. We kept the feature combinations selected by at least three of these techniques, yielding 40 transformed features. We then rebuilt the eight models with these 40 transformed features. We repeated this approach with the 50 features from prior research, with the results shown in Tables 4-6 and 4-7 under FT w/FS.

We then determined whether PCA could reduce the 1,326 features to “n” components, mixtures, or combinations of variables that captured the maximum variance. By using a cumulative scree plot, we identified 150 components that captured 79.9% of the variance (see Figure 4-1) from the 26 identified features and rebuilt the models with the components. We repeated this approach on the 50 features from prior research, identifying 300 components that captured 81.2% of the variance (see Figure 4-2) with the results shown in Tables 4-6 and 4-7 under FT w/PCA.

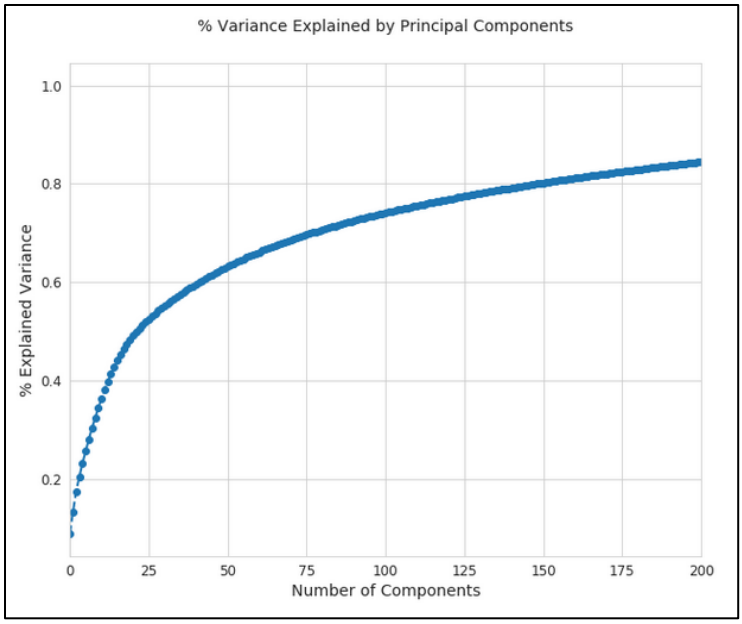


Fig. 4-1. 150 components are created from 26 identified webpage content features

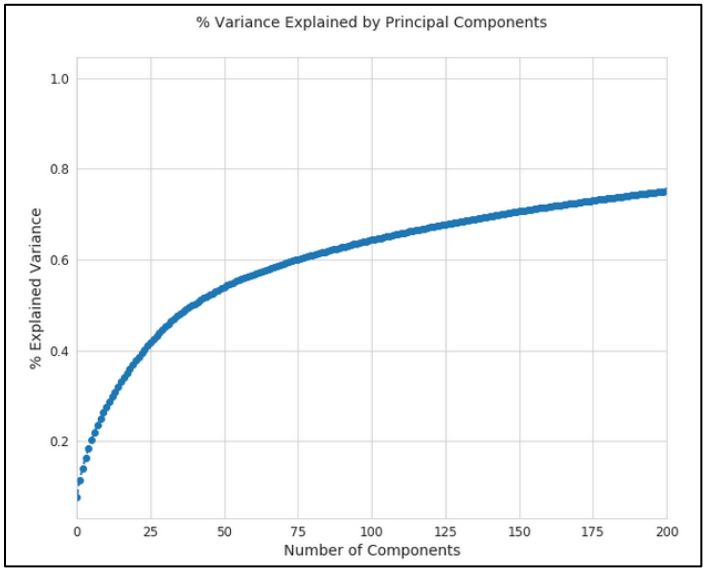


Fig. 4-2. 300 components are created from 50 identified webpage content features

Table 4-6.
Model Performance (50 Webpage Content Features from Prior Research / 26 Identified Webpage Content Features) with Feature Transformation

Model Performance (50 Webpage Content Features from Prior Research / 26 Identified Webpage Content Features) with Feature Transformation										
	FT w/FS					***FT w/PCA***				
Model	***FPR***	***FNR***	***ACC***	***AUC***	***MCC***	***FPR***	***FNR***	***ACC***	***AUC***	***MCC***
KNN	0.0079/ **0.0064**	0.8702/ 0.8768	0.8647/ 0.8650	0.5609/ 0.5583	0.2720/ 0.2719	0.0864/ 0.0864	0.1247/ 0.1306	0.9078/ 0.9070	0.8943/ 0.8914	0.6957/ 0.6916
LR	0.1170/ 0.1129	0.1763/ 0.1780	0.8742/ 0.8774	0.8533/ 0.8544	0.6038/ 0.6095	0.0874/ 0.0906	0.2063/ 0.1455	0.8949/ 0.9012	0.8531/ 0.8818	0.6363/ 0.6730
RF	0.0822/ 0.0835	0.1189/ 0.1206	**0.9123**/ 0.9109	**0.8993**/ 0.8978	**0.7081**/ 0.7043	**0.0804**/ 0.0824	0.1247/ 0.1239	**0.9130**/ 0.9114	0.8974/ 0.8968	**0.7078**/ 0.7043
AB	0.0828/ 0.0827	0.1805/ 0.2179	0.9027/ 0.8973	0.8683/ 0.8496	0.6639/ 0.6376	0.0835/ 0.0840	0.1697/ 0.1522	0.9037/ 0.9059	0.8733/ 0.8818	0.6701/ 0.6814
GB	0.0805/ 0.0801	0.1555/ 0.1589	0.9083/ 0.9082	0.8819/ 0.8804	0.6861/ 0.6847	0.0809/ 0.0814	0.1464/ 0.1489	0.9093/ 0.9086	0.8862/ 0.8848	0.6916/ 0.6890
ET	0.0825/ 0.0838	0.1181/ 0.1181	0.9121/ 0.9110	0.8996/ 0.8990	0.7080/ 0.7054	0.0809/ 0.0834	0.1247/ 0.1189	0.9125/ 0.9113	0.8971/ 0.8988	0.7067/ 0.7057
BC	0.0825/ 0.0844	0.1198/ 0.1198	0.9119/ 0.9103	0.8988/ 0.8978	0.7069/ 0.7031	0.0825/ 0.0841	0.1222/ 0.1181	0.9115/ 0.9108	0.8975/ 0.8988	0.7052/ 0.7049
NN	0.1180/ 0.1139	**0.1031**/ 0.1272	0.8841/ 0.8840	0.8894/ 0.8793	0.6535/ 0.6435	0.0913/ 0.0935	0.1106/ **0.1089**	0.9057/ 0.9042	**0.8989**/ 0.8987	0.6960/ 0.6930

Table 4-7.
Model Performance (50 Webpage Content Features from Prior Research / 26 Identified Webpage Content Features) with Feature Transformation (cont.)

Model Performance (50 Webpage Content Features from Prior Research / 26 Identified Webpage Content Features) with Feature Transformation				
	FT w/FS		***FT w/PCA***	
Model	***Prec***	***Rec***	***Prec***	***Rec***
KNN	0.7393/ **0.7668**	0.1297/ 0.1231	0.6368/ 0.6352	0.8752/ 0.8693
LR	0.5493/ 0.5575	0.8236/ 0.8219	0.6111/ 0.6201	0.7936/ 0.8544
RF	0.6496/ 0.6456	0.8810/ 0.8793	0.6534/ 0.6480	0.8752/ 0.8760
AB	0.6314/ 0.6208	0.8194/ 0.7820	0.6324/ 0.6360	0.8302/ 0.8477
GB	0.6448/ 0.6451	0.8444/ 0.8410	0.6460/ 0.6442	0.8535/ 0.8510
ET	0.6491/ 0.6455	0.8818/ 0.8818	**0.6517**/ 0.6465	0.8752/ 0.8810
BC	0.6486/ 0.6435	0.8801/ 0.8801	0.6480/ 0.6447	0.8777/ 0.8818
NN	0.5682/ 0.5701	**0.8968**/ 0.8727	0.6277/ 0.6226	0.8893/ **0.8910**

The MCC improved only in the LR model when comparing the models built with the prior 50 features to the models built with the 26 features identified in the feature transformation with feature selection case. Additionally, in this case, the average MCC decreased from 0.6253 to 0.6201 when using the 26 features instead of the 50 features from prior research. Also, when considering the impact of feature transformation with feature selection on the 26 features compared to no feature selection, feature transformation with feature selection reduced the average MCC from 0.6865 to 0.6201. When we applied PCA to the transformed features, the MCC only increased in two of the models – LR and AB – when considering the 26 features identified in our research rather than the 50 previously studied features, but the average MCC increased from 0.6887 to 0.6929. When considering the impact of feature transformation with PCA on the 26 features compared to no feature transformation, feature transformation with PCA increased the average MCC from 0.6865 to 0.6929.

Although the features we identified did not greatly improve malicious website detection (there was only an increase of 0.005 in the average MCC overall), the features we identified did improve malicious website detection with 48% fewer features in the scenarios without feature transformation and in the feature transformation with PCA.

4.7.3 RQ3: Do our Results Change with No-sampling, Under-sampling, and Over-sampling Scenarios?

RQ3 addressed the sensitivity of our approach and the impact of dataset imbalance. Sampling is especially important in malicious website classification because researchers (ourselves included) use datasets that are imbalanced. There is neither a standard that dictates when to perform sampling nor a standard of how much of an

imbalance between malicious and non-malicious should be used to train and test malicious website detection models. Hence, exploring whether sampling affects the results is worthwhile. We compared the feature rankings and the overall performance of our classifiers.

In the ranking of the 26 features identified in our research, the top three were consistent in the three-sampling scenarios. Although these were only three consistent rankings, they accounted for approximately 50% of total feature importance. We did however, observe some change in the MCC over the sampling scenarios with MCCs of 0.6865, 0.6144, and 0.5910, respectively, in the no-sampling, over-sampling and under-sampling scenarios. In case of the 50 features gathered from prior research, the only ranking that was consistent was the first, with the MCCs for the respective sampling scenarios being 0.6812, 0.6015, and 0.5842, respectively, for the no-sampling, over-sampling, and under-sampling cases.

The answer to RQ3 was mixed. We observed that rankings were not consistent, though the rankings of the features with the highest importance demonstrated consistency. The MCCs, however, were more consistent across the sampling scenarios.

4.7.4 RQ4: Does Hyperparameter Tuning and Cross-Validation Improve our Results?

We performed hyperparameter tuning and cross-validation to explore their effects and to provide assurance that results in Tables 4-4, 4-5, 4-6, and 4-7 were comparable to the tuned and cross-validated results. In each scenario – no-sampling, over-sampling, under-sampling, feature transformation with feature selection, and feature transformation with PCA – we chose the best performing model, built from the 26 features, and

proceeded to tune the parameters and cross-validate. We leveraged a decision tree classifier as the base estimator and StratifiedKFold [190] for 10-fold cross-validation. None of the MCC results from the five models improved, with the average MCC only decreasing from 0.7051 to 0.6999, suggesting that using the default parameters in [29] for our models in Tables 4-4, 4-5, 4-6, and 4-7 was sufficient.

We also needed to ensure that our results were not dependent on the 80:20 split of train/test data. To do this, we repeated our approach, as well as parameter tuning and cross-validation of the best models, but on a 70:30 split of training to test data instead of 80:20. Tuning and cross-validation did not improve any of the models for the 70:30 split, but the average MCC decreased from 0.7043 to 0.6908. Without tuning and cross-validation, the average MCC was 0.7043 and 0.7051, respectively, with the 70:30 and 80:20 split. With tuning and cross-validation the average MCC was 0.6908 and 0.6999, respectively, with the 70:30 and 80:20 split. The results were similar, suggesting that we were not dependent on the train/test split. Full results are shown in Table 4-8 below.

Table 4-8.
Cross-Validation and Hyperparameter Tuning Slightly Improved Webpage Content Models

Cross-Validation and Hyperparameter Tuning Webpage Content Models			
Model	***Scenario - Split***	***MCC***	***Scoring Metric***
ET	No-sampling - 70:30	0.6880	recall macro
ET	Over-sampling - 70:30	0.6865	accuracy
ET	Under-sampling - 70:30	0.6763	recall macro
RF	FT w/ FS - 70:30	0.6984	balanced accuracy
ET	FT w/ PCA - 70:30	0.6906	balanced accuracy
RF	No-sampling - 80:20	0.7029	precision macro
RF	Over-sampling - 80:20	**0.7126**	balanced accuracy
RF	Under-sampling - 80:20	0.6808	accuracy
ET	FT w./FS - 80:20	0.6977	balanced accuracy
ET	FT w/ PCA - 80:20	0.7055	balanced accuracy

Although we tuned our model hyperparameters and cross-validated, we did not see improvement of the average MCC in the 80:20 and 70:30 cases.

4.8 Conclusion

This chapter included a comprehensive evaluation of webpage content features to demonstrate the potential of using webpage content features alone to detect malicious websites and to determine whether new, unstudied webpage content features could improve malicious website detection. We analyzed webpage content features from 5,931 malicious websites and from 34,778 benign websites. Malicious websites were identified by Cisco Talos, while benign websites were gathered from the Alexa Top 1M. We collected 17,746 webpage content features from these websites and identified 26 for further analysis, of which, 17, to the best of our knowledge, were new. We built and evaluated eight models and ensured that our results were not greatly impacted by our dataset imbalance by performing no-sampling, over-sampling, and under-sampling

scenarios. We further demonstrated consistency in our results by performing feature transformations, rebuilding the models, and comparing results.

We compared the results from models built with the 26 features identified by our approach with results from models built with 50 features gathered from prior research. Additionally, we observed that the relative importance of the features decreased gradually with rank except for the first, and in some cases the second, ranked feature. The average MCC for the 26 features identified from our research was slightly higher than the average MCC for the 50 previously studied features, but used roughly half of the features. When considering the 26 selected features, feature transformation with feature selection decreased the MCC, while feature transformation with PCA increased the average MCC. Our results indicated the existence of a broader set of webpage content features that can be used for malicious website detection than those features commonly studied by previous researchers.

Chapter 5: URL Features Analysis

5.1 Introduction

A URL specifies the internet location of a resource – most commonly a website. The URL allows for the retrieval of documents, webpages, and other files across the internet and can do so with or without the actual IP address. Although website URLs have legitimate uses, they also enable many threats on the internet. URLs can point to phishing websites, to websites that conduct drive-by downloads, or to C2 websites, for example. Prior research has noted that malicious URLs often have a distinct structure when compared to benign URLs. Thus, the structure of the URL has been explored for malicious website detection and we conduct an additional analysis in this chapter. Our contributions are detailed below.

- We demonstrated the potential of using only URL features as a means to detect malicious websites on a dataset consisting of multiple types of threats.
- Among the 41 features we identified, we introduced five features focused on the number of English words of a given length that had not been studied in terms of detecting malicious websites.
- We observed that counts of the letters in the English alphabet account for an average of 35% of feature importance across our sampling scenarios.
- When considering the 41 selected features, feature transformation with feature selection and PCA decreased the MCC compared to the no-sampling scenario with no feature transformation.

5.2 Related Research

In this section, we summarize previous research and the use of URL features for malicious website detection. URL features have been used in many works and this section groups together works that have commonalities.

Early research by [34]-[37] and [40] leveraged a "bag-of-words" approach that separates the URL based on special characters ("=," "." "?" etc.) and examined the resulting tokens. In addition to using special characters as delimiters, researchers have used the presence or counts of specific special characters as features for malicious website detection [24], [40], [43]-[44], [46]-[50], [64], [81], [89], [116]. The "." character is heavily used because it separates domain names including TLDs and subdomains in the URL. Another feature, the URL length, is one of the features most prevalently leveraged in prior research. Prior researchers noticed that malicious URLs are typically longer (or shorter) and hence the `url length` has been used to detect malicious websites [40], [44], [47], [49], [51], [64], [81], [89].

Some methods for detecting malicious URLs also take the structure of the URL (protocol, host, subdomain, domain, path, query parameters) into consideration and were demonstrated in [40], [42], [47], [81], [86], [91], [117]. Although this approach facilitates the extraction of more features, it presents a problem in potential test sets in that benign sets, such as ours, are usually the home pages of the domain, while test sets for malicious websites may have multiple subdomains, different paths, and varying query parameters. Such features would not be applicable to benign websites. Furthermore, to create a benign test set of URLs that have paths would require use of a web crawler or similar method that could introduce influence into the study. Another structural property

used in prior research is the presence of an IP address or a port number in the URL and has been studied by [24], [42]-[46], [48]-[50], [64], [81], and [116].

URL characteristics have been used to detect bots and malicious traffic. Kheir et al. [55] detected C2 communications through the clustering of URLs generated by malware. Yadav et al. [209] developed a method to detect DNS "fluxing" by examining bigrams in algorithmically generated URLs. Researchers [44] used the presence of multiple TLDs, which can be expressed as n-grams, in the domain as another feature. Huang et al. [160] proposed a method for dynamically extracting patterns from URLs (as opposed to n-grams) for malicious URL detection. Daeef et al. [95] used n-grams in conjunction with separating URLs into host, path, and query segments. Verma and Das [56] also used n-grams and extracted overlapping sequences of consecutive characters in the ranges of $N = 1$ to $N = 10$ and discussed the speed of their n-gram feature extractor. Authors [52] distinguished between algorithmically generated domains (AGDs) and human generated domains (HGDs), using `url length`, vowels, consonants, and digits, while [116] used the ratio of the number of specific characters over the total `url length`, among others factors, in their set of 41 features.

Whether using a "bag-of-words" approach, a structural approach, or a length-and-character approach, n-grams present in a URL have played a key role in the detection of malicious websites. As such, we used n-grams as the main set of features in our malicious website detection experiments. We extracted features from previous research and added several new features from n-grams based on English words, TLDs, file extensions, and well-known ports, with the goal of identifying new URL-based features for malicious website detection and building capable detection models.

5.3 Research Questions

We created three of our research questions with the aim of exploring the effectiveness of our approach and at assessing the URL features we identified as features for malicious website detection. These three research questions focus on using features derived from the URL as the sole source of features for the detection of malicious websites.

5.3.1 Research Question 1

Previous research applied several techniques and features to the analysis and detection of malicious website URLs. Currently, no definitive list of URL features exists, though certain features have been used extensively in prior research. Given that URLs have been analyzed in many ways and that diverse features have already been used in malicious website detection, we postulated that additional features might be relevant for malicious website detection. We hypothesized that our approach, which considered 28,162 features, many of which had never been studied for malicious website detection, would identify new features of importance in the detection of malicious websites. RQ1 is stated below.

RQ1: How do the features identified compare with prior research?

5.3.2 Research Question 3

This second question (third of our 13 research questions) focused on the consistency of our approach by investigating if our results changed across three sampling scenarios: no-sampling, under-sampling, and over-sampling. Class imbalances between the benign and malicious datasets are common in security research. With this question,

we analyzed whether our model performance changed across three sampling scenarios. We stated RQ3 as follows:

RQ3: Do our results change with no-sampling, under-sampling, and over-sampling scenarios?

5.3.3 Research Question 4

With this research question, we explored the use of hyperparameter tuning and cross-validation on our results. These techniques have the potential to improve our models and aided us in understanding how much our models could be improved (if at all). These additional methods gave our results more credence. RQ4 is as follows:

RQ4: Does hyperparameter tuning and cross-validation improve our results?

5.4 Feature Consideration

URLs have several characteristics we can extract. URL features are created by examining properties and patterns in the URL strings. In our approach, we leveraged features from previous research and expanded our study to include features not previously used in previous research. For a full list of URL features used in this study, please refer to Appendix A.

5.4.1 N-gram Approach

We took an n-gram approach that looked for specific n-grams in the URL. The n-gram approach is inspired by the “bag-of-words” approach used by many authors to detect phishing URLs and is influenced by the fact that n-grams have been used in several ways to successfully detect malicious websites. Our n-grams, however, were shaped by the n-grams used in previous research and extended to include additional relevant n-grams. Our first set of n-grams consisted of all English words. We used [210]

as our source of words. Specifically, we looked for the counts of all words from the dictionary with a length of four letters or more. We chose a word length of four letters in order to filter out simple connecting words such as “the” or “and.” We extracted the specific word, counted the number of times it was present in the URL, and counted the number of unique words of a given length that were present in that URL. For example, in the URL `homedepot.com`, we would identify the words `home` and `depot` resulting in a value of one for word_count_4 and word_count_5. The next n-grams we extracted were the presence of TLDs like `.com`, `.net`, or `.us`, motivated by the fact that multiple TLDs have been used in malicious website detection. We used the list of TLD names from the Internet Corporation for Assigned Names and Numbers and the list of file extensions from [211]. The third set of n-grams we extracted was the presence of file extensions. URLs can point to files (`.exe`, `.zip`, etc.), with previous researchers focusing on whether a URL points to an executable file.

5.4.2 **Character Distributions**

Character distributions and the number of certain special characters (the “.” and the “-,” for example) are known features for malicious website detection. In addition to special characters, regular characters such as consonants, vowels, and digits also have been used to detect malicious websites. Typically, different ratios of characters are grouped together to detect bot URLs and URLs generated by DGAs. We extended these approaches by capturing the total number of digits, vowels, consonants, and special characters, as well as the counts for each type of character.

5.4.3 Specific Features

Lastly, we collected those features that are specific to the URL structure. We first checked for the presence of an IP address in the URL, since an IP address substituted for a hostname is a known technique for obfuscating a malicious URL. In addition to IP addresses, we also looked for the presence of port numbers. If we found a port number in a URL, we recorded it and checked to see whether it is a well-known port number. Well-known port numbers include 22 for `ssh`, 25 for `smtp`, and 53 for `DNS`, among others.

5.5 Learning, Feature Selection, and Sampling Techniques in URL Analysis

5.5.1 Feature Selection

After initial collection of the 28,162 URL features, we analyzed which features had a strong association with the dependent variable (i.e., whether the website was malicious) and eliminated any redundant features (those that had no relationship or a weak relationship with the dependent variable). We removed those features that had the same value at least 95% of the time, thereby eliminating 28,121 features and resulting in a final set of 41 features.

5.5.2 Machine Learning Models, Sampling, and Feature Transformation

We evaluated the feature set against eight different supervised classifiers discussed in Section 3.4.1 and recorded their performance metrics. For all models, we split training and testing data using an 80:20 ratio, a common train/test split for data. Our overall dataset was imbalanced: we had 39,877 benign websites and 6,894 malicious websites. To address this and to ensure that any results were not the product of our benign-to-malicious website ratio, we trained the models using different samples of the benign and malicious datasets in the three sampling scenarios discussed in Section 3.4.3.

For the no-sampling scenario, we used 31,892 and 7,985 benign websites and 5,525 and 1,369 malicious websites, respectively, in our training and testing sets. Under-sampling resulted in 5,525 malicious websites and 5,525 benign websites in the under-sampled training set. Over-sampling with the SMOTE technique [186] from [187] produced a balanced training set with 31,892 benign websites and 31,892 malicious websites. We also built models with transformed features created from the process in Section 3.4.3.

The websites used in the testing dataset remained consistent across all models and sampling scenarios for the training data so that we could compare model results and identify whether a sampling technique led to a better model. We ensured that there was no overlap between training and testing datasets.

5.6 Results

5.6.1 RQ1: How do the Features Identified Compare with Prior Research?

With RQ1, we investigated whether or not our approach identified previously studied URL features as important. To do so, we leveraged our four ensemble methods (RF, AB, ET, and BC), all of which captured the notion of feature importance – a normalized metric between 0 and 1.0 for each respective feature. The top 41 features are shown below in Table 5-1, along with their respective rank in the no-sampling, over-sampling, and under-sampling cases and with their respective header field. The white rows indicate features previously studied in prior research for the identification of malicious websites, while the shaded rows are features that, to our knowledge, are new. Rank and importance are separated by a ":" character.

Table 5-1.
The Top Seven URL Features Had Consistent Rank

41 Identified URL Features Ranked			
URL Feature	***No-sampling***	***Over-sampling***	***Under-sampling***
Total Extensions in URL	1 : 0.1978	1 : 0.1874	2 : 0.1657
URL Length	2 : 0.1815	2 : 0.1133	1 : 0.2105
Count of '.' character	3 : 0.0726	3 : 0.0685	3 : 0.0796
Count of 'w' character	4 : 0.0699	4 : 0.0496	4 : 0.0503
number of consonants in the URL	5 : 0.0520	6 : 0.0386	5 : 0.0475
number of digits in the URL	6 : 0.0430	5 : 0.0485	6 : 0.0408
Total TLDs in URL	7 : 0.0343	8 : 0.0335	7 : 0.0398
Count of 'z' character	8 : 0.0298	14 : 0.023	8 : 0.0319
Count of .com in URL	9 : 0.0235	21 : 0.016	11 : 0.0200
Count of 4-character words	10 : 0.0221	18 : 0.0201	9 : 0.02919
Number of vowels in the URL	11 : 0.0197	23 : 0.0146	10 : 0.0249
Count of 'i' character	12 : 0.0156	7 : 0.03385	13 : 0.0186
Count of 'b' character	13 : 0.0132	19 : 0.0186	12 : 0.0193
Count of 'y' character	14 : 0.0131	20 : 0.0168	18 : 0.0112
Count of 'l' character	15 : 0.0128	12 : 0.0273	15 : 0.0127
Count of 'm' character	16 : 0.0120	27 : 0.0082	17 : 0.0117
Count of 'o' character	17 : 0.0119	15 : 0.0223	14 : 0.0137
Count of 't' character	18 : 0.0114	10 : 0.0278	20 : 0.0110
Count of 'p' character	19 : 0.0113	16 : 0.0208	32 : 0.0060
Count of 'n' character	20 : 0.0111	30 : 0.0060	16 : 0.0125
Count of 'x' character	21 : 0.0109	33 : 0.0043	23 : 0.0107
Count of 'f' character	22 : 0.0108	25 : 0.0112	22 : 0.0109
Count of 'r' character	23 : 0.0106	11 : 0.0277	21 : 0.0110
Count of 'h' character	24 : 0.0098	26 : 0.0107	24 : 0.0098
Count of 'g' character	25 : 0.0094	35 : 0.0038	25 : 0.0091
Count of 'e' character	26 : 0.0084	22 : 0.0158	19 : 0.0110
Count of .i in URL	27 : 0.0080	38 : 0.0026	26 : 0.0079
Count of 'j' character	28 : 0.0077	40 : 0.0019	41 : 0.0025
Count of 's' character	29 : 0.0069	9 : 0.02914	31 : 0.0061
Count of .net in URL	30 : 0.0067	28 : 0.0080	28 : 0.0071
Count of 'c' character	31 : 0.0064	24 : 0.0120	27 : 0.0074
Count of 'a' character	32 : 0.0060	29 : 0.0061	29 : 0.0065
Count of 'u' character	33 : 0.0057	17 : 0.0207	30 : 0.0064
Count of 5-character words	34 : 0.0056	31 : 0.0060	33 : 0.0058
Count of 'd' character	35 : 0.0055	13 : 0.0236	35 : 0.0053
Count of 6-character words	36 : 0.0046	32 : 0.0050	34 : 0.0055
Count of 'k' character	37 : 0.0043	34 : 0.0041	36 : 0.0043
Count of 'v' character	38 : 0.0033	37 : 0.0031	39 : 0.0033
Count of 7-character words	39 : 0.0033	36 : 0.0031	38 : 0.0033
Count of '-' character	40 : 0.0031	39 : 0.0025	37 : 0.0034
Count of 8-character words	41 : 0.0025	41 : 0.0018	40 : 0.0030

5.6.1.1 Features Identified in Previous Research

In our list of 41 features, 36 had been used in prior research, while the remainder were introduced in our study. The `url-length` consistently ranks highly and has been used by nearly all research that uses any URL features. The `number of file extensions` in the URL also ranked highly. While no research of which we are aware has used this feature explicitly, some scholars have examined whether or not the URL

points to a specific type of file, so we included this feature as a prior feature. Counts of the special characters "." and "-" also appear in our list and, like `url-length`, are very commonly studied features. However, we found it surprising that the count of "-" was ranked so low (40, 39, and 37, respectively, in the no-sampling, over-sampling, and under-sampling cases), given its frequent use in prior research. Distributions of vowels, digits, and consonants have been used to identify C2 websites, in particular, and appeared on our list. We also observed that the count of every letter with the exception of "q" ranked on our list.

5.6.1.2 New Features Identified

Thirty-six of the 41 features were identified in prior research or were closely related enough to be considered part of prior research. However, our research identified five new features that can facilitate malicious website detection. These five features all represent the number of English words of a given length in the URL. Certain words have been associated with phishing websites [34], though, to our knowledge, no approaches have incorporated the length of words present in the URL. We also observed that all letters of the alphabet contributed to the detection of the malicious website except for the letter "q."

5.6.1.3 Features Ranking Analysis

We used ensemble methods (RF, AB, ET, and GB) to understand feature importance. Table 5-1 presents the 41 features, along with their average respective rank and importance with no-sampling, under-sampling, and over-sampling using the four ensemble methods [182]. We observed that the first two features were consistently ranked as the top two features both with and without sampling and had an importance

much higher than the remaining 36 features. Specifically, `url-length` and the `number of file extensions` had a combined importance of 0.3793, 0.3008, and 0.376, respectively, in the no-, over-, and under-sampling cases for the 41 features. We also observed that the feature rank and importance were similar when considering no-sampling, over-sampling, or under-sampling, with the top six features being the same (but in different order) in the various sampling scenarios. These six features accounted for 0.6170, 0.5063, and 0.5947, respectively, of cumulative importance. In the previous section, we noted that counts of specified characters, with the exception of the letter "q," appeared in our list. When we summed the respective importances of the counts of letters, we got cumulative importances of 0.3191, 0.4263, and 0.3044, respectively, in the no-sampling, over-sampling, and under-sampling cases.

URL features have been extensively studied in prior research. Thus, we only identified five new features, all of which centered around the counts of words of specific lengths that were present in the URL. Our approach identified features from prior research, reinforcing the importance of character counts for malicious website detection.

5.6.2 RQ3: Do our Results Change with No-sampling, Under-sampling, and Over-sampling scenarios?

We then investigated model performance for the test dataset when using the no-sampling, under-sampling, and over-sampling scenarios. Tables 5-2 and 5-3 provide FPR, FNR, ACC, AUC, MCC, Prec, and Rec for the three sampling. We focused on the average MCC of all models to motivate the discussion of our results. Averaging the MCC also provided an overall idea of how well the models performed, taking into account the respective performances of each model into a single metric.

Table 5-2.
URL Features Produced High Detection Metrics with 41 Identified URL Features in Sampling Scenarios

Model Performance (41 Identified URL Features) in Sampling Scenarios															
	No-sampling					***Over-sampling***					***Under-sampling***				
Model	***FPR***	***FNR***	***ACC***	***AUC***	***MCC***	***FPR***	***FNR***	***ACC***	***AUC***	***MCC***	***FPR***	***FNR***	***ACC***	***AUC***	***MCC***
KNN	**0.0032**	0.2652	0.9584	0.8658	0.8254	0.0735	0.1359	0.9174	0.8953	0.7135	**0.0316**	0.1789	**0.9469**	0.8947	**0.7878**
LR	0.0144	0.2454	0.9518	0.8701	0.7973	0.0705	**0.1169**	0.9227	**0.9063**	0.7330	0.0731	0.1242	0.9194	0.9013	0.7223
RF	0.0073	0.1994	**0.9646**	0.8967	**0.8527**	0.0111	0.1928	0.9623	0.8980	0.8432	0.0496	0.1293	0.9387	0.9106	0.7731
AB	0.0148	0.2564	0.9499	0.8644	0.7887	0.0476	0.1855	0.9322	0.8834	0.7397	0.0923	**0.1227**	0.9033	0.8925	0.6846
GB	0.0124	0.2199	0.9572	0.8839	0.8212	0.0279	0.1885	0.9486	0.8918	0.7921	0.0585	0.1264	0.9316	0.9076	0.7536
ET	0.0074	0.2023	0.9640	0.8951	0.8503	**0.0088**	0.1928	**0.9643**	0.8992	**0.8515**	0.0501	**0.1227**	0.9393	**0.9136**	0.7763
BC	0.0101	0.2053	0.9613	0.8923	0.8386	0.0103	0.1987	0.9622	0.8955	0.8424	0.0554	0.1315	0.9335	0.9066	0.7575
NN	0.0247	**0.1585**	0.9557	**0.9084**	0.8218	0.0397	0.1651	0.9450	0.8976	0.7745	0.0909	0.1191	0.9050	0.8950	0.6898

Table 5-3.
URL Features Produced High Detection Metrics with 41 Identified URL Features in Sampling Scenarios (cont.)

Model Performance (41 Identified URL Features) in Sampling Scenarios						
	No-sampling		***Over-sampling***		***Under-sampling***	
Model	***Prec***	***Rec***	***Prec***	***Rec***	***Prec***	***Rec***
KNN	**0.9748**	0.7348	0.6683	0.8641	**0.8168**	0.8210
LR	0.8998	0.7545	0.6822	**0.8831**	0.6724	0.8758
RF	0.9497	0.8005	0.9254	0.8071	0.7506	0.8707
AB	0.8961	0.7436	0.7458	0.8144	0.6197	0.8772
GB	0.9151	0.7801	0.8328	0.8115	0.7191	0.8736
ET	0.9487	0.7976	**0.9404**	0.8071	0.7501	0.8772
Bag	0.9307	0.7947	0.9304	0.8013	0.7290	0.8685
NN	0.8539	**0.8414**	0.7828	0.8349	0.6242	**0.8809**

Without sampling, the average MCC was 0.8245. When over-sampling, the average MCC was 0.7862. In the under-sampling case, the average MCC was 0.7431. Throughout various sampling scenarios, this method showed promise for malicious website detection.

We also explored the model performance with transformed features created from the process in Section 3.4.3. We performed feature-transformation for the 41 features in order to determine whether we could improve upon the performance (increase the average MCC). We transformed the original 41 features into 3,321 features. We then performed feature selection on the 3,321 features with the four different techniques from Section 3.4.3, resulting in 33 transformed features.

We also attempted to determine whether PCA could reduce the transformed features to a group of components that captured the maximum variance among the data. Using a cumulative scree plot, we found that 110 components captured 80.65% of the variance in the dataset for 41 identified features (see Figure 5-1). We used these 110 components in our subsequent analyses to assess their performance detecting malicious websites. Results are shown in Tables 5-4 and 5-5.

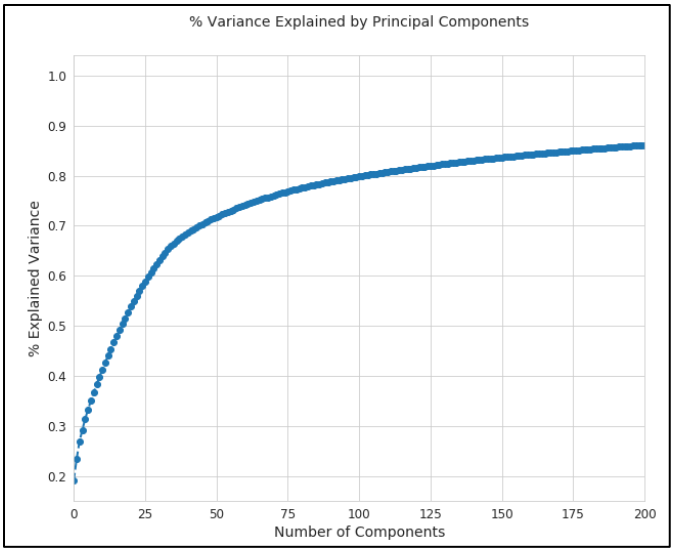


Fig. 5-1. 110 components are created from 41 URL features

Table 5-4.
URL Features Produced High Detection Metrics with 41 identified URL Features in Feature Transformation Scenarios

Model Performance (41 Identified URL Features) with Feature Transformation										
Model	***FT w/FS***					***FT w/PCA***				
	FPR	***FNR***	***ACC***	***AUC***	***MCC***	***FPR***	***FNR***	***ACC***	***AUC***	***MCC***
KNN	0.0169	**0.2220**	0.9531	0.8805	0.8044	**0.0020**	0.3908	0.9411	0.8036	0.7467
LR	0.0164	0.2885	0.9438	0.8475	0.7617	0.0140	0.2257	**0.9550**	0.8801	**0.8117**
RF	0.0138	0.2301	0.9546	0.8781	0.8097	0.0113	0.3185	0.9438	0.8351	0.7591
AB	0.0170	0.3112	0.9399	0.8359	0.7433	0.0198	0.2827	0.9417	0.8488	0.7536
GB	0.0160	0.2498	0.9498	0.8671	0.7888	0.0124	0.2535	0.9523	0.8671	0.7990
ET	0.0160	0.2264	0.9532	0.8788	0.8044	0.0055	0.3740	0.9406	0.8102	0.7434
BC	0.0148	0.2243	0.9546	0.8805	0.8101	0.0132	0.2776	0.9482	0.8546	0.7801
NN	**0.0104**	0.2243	**0.9583**	**0.8827**	**0.8254**	0.0232	**0.1812**	0.9537	**0.8978**	0.8114

Table 5-5.
URL Features Produced High Detection Metrics with 41 Identified URL Features in Feature Transformation Scenarios (cont.)

Model Performance (41 Identified URL Features) with Feature Transformation				
	FT w/ FS		***FT w/ PCA***	
Model	***Prec***	***Rec***	***Prec***	***Rec***
KNN	0.8875	**0.7779**	**0.9811**	0.6092
LR	0.8814	0.7114	0.9044	0.7742
RF	0.9054	0.7699	0.9120	0.6815
AB	0.8739	0.6888	0.8614	0.7173
GB	0.8891	0.7501	0.9116	0.7465
ET	0.8921	0.7735	0.9511	0.6260
Bag	0.9000	0.7757	0.9040	0.7224
NN	**0.9275**	0.7757	0.8583	**0.8188**

For feature transformation with feature selection, the MCC was 0.7934. With feature transformation with PCA, the average MCC was 0.7756. Both show promise that our approach can detect malicious websites however, we found that feature transformation with feature selection and PCA both worsened the average MCC when compared to the no-sampling case.

We next used dataset sampling to investigate the consistency of our approach and its robustness over class imbalance. The MCCs were 0.8245, 0.7862, and 0.7431 for 41 features identified in our approach, showing slight variation from the no-sampling scenario, where the MCC was 0.8245 , to the under-sampling case, where the MCC was 0.7431. The MCC for the over-sampling case was 0.7862. Nevertheless, all three scenarios still showed promise for malicious website detection.

We observed slight disparities in model performance across the sampling scenarios and observed that the set of the top six most important features were consistent, accounting for 0.6170, 0.5063, and 0.5947, respectively, of cumulative importance in the no-sampling, under-sampling, and over-sampling scenarios.

5.6.3 RQ4: Does Hyperparameter Tuning and Cross-Validation Improve our Results?

In this step, we investigated the impact of hyperparameter tuning and cross-validation on our results. We performed hyperparameter tuning and cross-validation on our dataset and re-evaluated our models with a 70:30 split of train to test data instead of the initial 80:20 split. Doing so ensured that our models were not overfit and that they had the potential to improve our models. Furthermore, this reinforced that our observations were not dependent on the initial 80:20 split of data.

We performed hyperparameter tuning and cross-validation on the best performing models in each scenario – no-sampling, over-sampling, under-sampling, feature transformation with feature selection, and feature transformation with PCA. In the 80:20 case, all five models improved, but the average MCC only increased from 0.8258 to 0.8343, suggesting consistency of the results in Tables 5-2, 5-3, 5-4, and 5-5 even when we tuned the parameters and performed cross-validation.

In the 70:30 case, tuning and cross-validation improved three of the five models for the 70:30 split, but the average MCC only increased from 0.8303 to 0.8399. Without tuning and cross-validation, the average MCC was 0.8303 and 0.8258, respectively, with the 70:30 and 80:20 splits. With tuning and cross-validation, the average MCC was 0.8399 and 0.8343, respectively, with the 70:30 and 80:20 splits. The small difference between results in the different splits suggested that we were not dependent on the train/test split. Results are shown in Table 5-6 below.

Table 5-6.
Cross-Validation and Hyperparameter Tuning Slightly Improved URL Models

Cross-Validation and Hyperparameter Tuning URL Models			
Model	***Scenario – Split***	***MCC***	***Scoring Metric***
ET	No-sampling - 70:30	**0.8655**	balanced accuracy
ET	Over-sampling - 70:30	0.8596	balanced accuracy
KNN	Under-sampling - 70:30	0.8236	precision weighted
NN	FT w/ FS - 70:30	0.8382	accuracy
LR	FT w/ PCA - 70:30	0.8124	recall weighted
RF	No-sampling - 80:20	0.8572	recall weighted
ET	Over-sampling - 80:20	0.8567	balanced accuracy
KNN	Under-sampling - 80:20	0.8163	precision weighted
NN	FS w/ FT - 80:20	0.8303	accuracy
LR	FS w/PCA - 80:20	0.8107	precision weighted

In Research Question 4 (three of 13), we determined two ways of validating our results. First, we performed hyperparameter tuning and cross-validation. Secondly, we

rebuilt and performed hyperparameter tuning and cross-validation on our eight models on a 70:30 split of the data. While we observed improvement after performing hyperparameter tuning and cross-validation, the improvement was small.

5.7 Conclusion

This chapter included a comprehensive evaluation of URL features for assessing whether additional URL features improve malicious website detection. We analyzed URL data from 6,894 malicious and 39,877 benign websites. We based our dataset of malicious websites on those identified by Cisco Talos and based our dataset of benign websites on the Alexa Top 1M. We collected 28,162 URL features from these websites and identified 41 for further analysis, including five newly identified features. We applied eight models and ensured robustness of our methodology by using three sampling scenarios – no-sampling, over-sampling and under-sampling.

Among the 41 features, the top six were consistent across the sampling scenarios and accounted for approximately 55% of the total feature importance. Also, we found the count of individual characters to be of importance in malicious website detection, accounting for an average importance value of approximately 35% over the three sampling scenarios. Lastly, we observed that counting the number of words of a given length may be an additional useful feature for malicious website detection.

Chapter 6: HTTP Features Analysis

6.1 Introduction

This chapter includes a comprehensive evaluation of an HTTP header-only approach to malicious website detection aimed at assessing whether additional HTTP header features can improve malicious website detection. Our contributions in this chapter are listed below.

- We demonstrated the potential of using HTTP header features alone as a means of detecting malicious websites.
- We introduced 11 new HTTP header features not previously considered as aiding in the detection of malicious websites.
- Eight of the 22 features, three of which were newly identified by our approach, ranked as the most important features and represented 80% of feature importance.
- The average MCC for the selected 22 features was better than the average MCC for the 11 previously studied features across our three sampling scenarios.
- We found that applying PCA to the 22 selected features improved malicious website detection.

6.2 Related Research

Features gathered over a session have been used to identify malicious websites and traffic. Authors [77] gathered features from HTTP `request`s and `response`s over a session and combined them with non-HTTP features in an attempt to detect malicious webpages. Authors [40], [49], and [81] took similar approaches, combining non-HTTP

features with specific metadata gathered from the interaction with a website. These approaches obtained detection rates of up to 96%. Researchers [212] used the `Content-Type` header as a means of distinguishing between different types of HTTP traffic, while [213] used HTTP application level features to distinguish different attack classes in traffic to their honeypot. These approaches demonstrate that specific HTTP features show potential for identifying malicious activity. However, prior researchers limited themselves to a small list of features or required additional non-HTTP features to achieve their performance metrics. With Phishmon, researchers [83] considered all HTTP headers as potential features, but used lengths of the respective headers.

Other approaches demonstrated that HTTP traffic generated by malware can be used to build signatures or fingerprints for detection. Authors [75]-[76] clustered the HTTP communications generated to and from HTTP-based malware on their testbed to create signatures. Brezo et al. [79] recorded HTTP traffic over a session and produced a list of the influential features for malicious traffic identification that consisted of TCP and HTTP features. They found the `Content-Length` header to be of importance. ARROW, by [78], generated signatures from redirect chains captured in HTTP traces. Kheir et al. [55] clustered HTTP traffic in order to classify the C2 communications. With BotHound, [82] found that malicious communications may have similar `User-Agent` strings in `request`s. Generating signatures or fingerprints for malicious HTTP communications was also used in [80], [162], [212], [214]-[215].

6.3 Research Questions

We created four research questions aimed at exploring the effectiveness of our approach and the header features we identified as features for malicious website

identification. With these four questions, we focused on using HTTP headers as the sole source of features for detection of malicious websites.

6.3.1 Research Question 1

With our first question, we compared the features identified in our approach with those gathered from prior research. Previous researchers used HTTP headers to detect malicious websites, but their use is limited. Furthermore, we did not consider session-based features, focusing instead on features extracted from the HTTP `responses` headers. While no definitive list of HTTP headers and features to use for malicious website detection exists, we created an approach designed to create such a list. Additionally, researchers have identified a select few HTTP header features for actual use in detecting malicious website detection. We hypothesized, however, that with our study of 672 features, many of which had never been explored for purposes of malicious website detection, we could identify new important features for the identification of malicious websites. To that end, we compared the header features identified by our approach with the header features used by previous authors. Research Question 1 is stated as follows:

RQ1: How do the features identified compare with prior research?

6.3.2 Research Question 2

With RQ2, we investigated whether the incorporation of these new features would improve malicious website detection. To accomplish this, we compared the MCCs with and without the additional 11 features identified in this work. We also built models with transformed features created from performing feature transformation techniques with

feature selection and with PCA, further comparing the respective MCCs. RQ2, then, is stated as follows:

RQ2: Do the additional features identified improve malicious website detection?

6.3.3 Research Question 3

We focused RQ3 on the consistency of our approach in sampling scenarios. In other words, we sought to determine whether our approach yielded consistent results in the cases of no-sampling, over-sampling, and under-sampling of our dataset. We, like other security researchers, worked with an imbalanced dataset. In this question, we analyzed how our models performed in the no-sampling, over-sampling, and under-sampling cases. RQ3 is stated as follows:

RQ3: Do our results change with no-sampling, under-sampling, and over-sampling scenarios?

6.3.4 Research Question 4

We used RQ4 to enable our exploration of additional tuning methods to our results. Although we were working with a single dataset, we intended to evaluate and include additional methods that would give our results more credence. Our fourth research question is stated as follows:

RQ4: Does hyperparameter tuning and cross-validation improve our results?

6.4 Feature Consideration

6.4.1 Extractable HTTP Features

Previous researchers have used HTTP traffic to identify and detect malicious websites, using two approaches. First, they applied HTTP traffic characterization, which involves the recording of HTTP traces and other features from known malicious websites

or from malware communicating with malicious websites. Although this approach has been used in bot detection, it requires additional dependencies and additional setup compared to the methodology we used for this research. HTTP traffic characterization also presents challenges of combining HTTP trace features with the other features studied in this research.

Previous researchers also employed a second method of exploiting HTTP traffic for detecting malicious websites – they used specific HTTP headers as part of a larger set of features. Although HTTP headers have been used in feature sets for malicious website detection, few researchers have emphasized HTTP headers, and none, to our knowledge, have used them outside of a "flow-based" method. We hypothesized that the lack of inclusion of HTTP headers in malicious website identification has resulted from the fact that HTTP header analysis is messy. First, while headers are specified in the HTTP specification, they also can be defined by users. Secondly, since the values in the HTTP headers can vary significantly, the process of researching and recording the possible values for headers is a tedious one. Lastly, in the process of collecting our HTTP headers, we observed that the values and the names of the headers frequently contained inconsistencies or misspellings that necessitated a pre-processing step. For example, we noticed the presence in our collection of two HTTP headers: `Accept-Encoding` and `Accept-encoding`. These headers are the same header but would be viewed as unique values without an additional pre-processing step because of the capitalization difference. There are other differences among headers as well, including misspellings of specific headers.

6.4.2 **HTTP Feature Collection**

An HTTP header is a key-value pair within an HTTP `request` or `response`, both of which may contain multiple headers. The example below shows the HTTP `request` headers generated during a web `request`. Bolded items are the header names (keys), while non-bold items are the corresponding values. We use the symbol "…" to indicate places where values were truncated due to length.

Accept	*/*
Accept-Encoding	gzip, deflate, br
Accept-Language	en-US,en;q=0.5
Authorization	SAPISIDHASH 1550122185_7937eb6...
Connection	keep-alive
Content-Length	3878
Content-Type	application/json
Cookie	YSC=b-ooV1KIyCk; VISITOR_INFO1…4555ce3QEAAAAdGxpcGn7+2...

For commonly used headers, please refer to MDN [206]. HTTP `response`s have a similar structure of key-value pairs. In this portion of the study, we performed a `GET request` to the selected websites, recording the headers present in the `response`. We only considered `response` headers.

HTTP feature collection took place in August 2018. We used the `Python requests` library [216] to make `GET requests` to the websites and collected HTTP features in the associated `response`. Upon receiving `responses`, we parsed and recorded the headers and values. The collection included features defined in the HTTP specification as well as custom headers defined by specific websites. We then examined the HTTP specification to determine whether the headers had a finite group of values. For example, the `Content-Security-Policy` header can have a finite group of directives in the header's value. Based on those directives, we collected additional features that captured whether the specific directive was present in the header. Another group of

features we gathered was defined by key-value pairs, which exist in the directives of certain headers. The example below shows a possible `Cache-Control` header.

```
Cache-Control: public, max-age=31536000
```

The public directive indicated that the `response` might be held in any cache, and the `max-age` directive was set to 31,536,000 seconds. Our method captured both of these features. Overall, data collection resulted in a total of 672 HTTP features.

6.5 Learning, Feature Selection, and Sampling Techniques in HTTP Header Analysis

6.5.1 Feature Selection

After collecting the 672 HTTP header features, we analyzed which of the features had strong association with the dependent variable (i.e., whether the website was malicious), eliminating any redundant features (i.e., those that had no relationship or a weak relationship with the dependent variable). We followed the process in Section 3.4.3. First, we removed the 399 features for which all the websites' HTTP `response` headers had the same value. Next, we removed features specific to our dataset by removing those that had the same value at least 95% of the time, thereby eliminating 245 features. We then evaluated the remaining 28 features to identify those features that had a high multicollinearity. Removing features with high multicollinearity ensured that we analyzed a set of independent features. Collinearity can be quantified by the VIF [177]. First, we determined the VIF values for each feature. We then iteratively identified features that had a VIF > 5, per [179]. Among our list of features with a VIF > 5, we determined which of the features had similar VIF values and high correlations to one another. We defined high correlation as having a correlation of greater than 0.7, as in

[180]. Among the highly correlated features with similar VIF values, we then removed those with the highest VIFs from our feature set, leaving us with final set of 22 features.

6.5.2 Machine Learning Models, Sampling, and Feature Transformation

We created two feature sets, the first of which included the 22 features identified by our approach and the second of which consisted of the 11 features identified in our approach that had also been studied in prior research. We evaluated the feature sets against eight different supervised classifiers discussed in Section 3.4.1, recording their performance metrics. For all models, we split training and testing data using an 80:20 split, which is a common train/test split. Our dataset was imbalanced, with 39,835 benign websites and 6,021 malicious websites. To address the imbalance and to ensure that our results were not a product of our benign-to-malicious split, we trained the models using different samples of the benign and malicious datasets. Specifically, we performed no-sampling, under-sampling, and over-sampling of the training dataset, which yielded three different training datasets that we used to evaluate models.

For no-sampling, we used 31,853 benign websites and 4,831 malicious websites in our training set. For under-sampling, we used the full set of malicious websites in the training set and selected a subsample from the benign websites to arrive at a training set of 4,831 malicious and benign websites respectively. For over-sampling, we derived a balanced training set of 31,853 benign websites and 31,853 malicious websites. The websites used in the testing set remained consistent across all models and sampling approaches so that we could compare results. Training and testing datasets did not overlap. Figure 3-3 provides a summary of the feature transformation and sampling techniques.

6.6 Results

6.6.1 RQ1: How do the Features Identified Compare with Prior Research?

With RQ1, we investigated whether our approach identified previously studied HTTP headers as important. To do so, we leveraged our four ensemble methods (RF, AB, ET, and BC), all of which captured the notion of feature importance – a normalized metric between 0 and 1.0 for each respective feature. Table 6-1 below displays the top 22 features, along with their respective rank in the no-sampling, over-sampling, and under-sampling cases and their respective header fields. That is, the "Feature" column specifies the header and, in some cases, specifies the value of that header. For example, `content-encoding gzip` specifies that the header `content-encoding` has a value `gzip`. The shaded rows are the new headers identified by our approach, while the unshaded rows indicate the headers gathered from previous scholarship. The ranking and respective importance values are separated by a ":" in the data columns. Table 6-2 shows the rankings from header features from prior work.

Table 6-1.
The Top 8 Identified HTTP Header Features Accounted for 81.62% of Importance

22 Identified HTTP Header Features Ranked			
Feature	***No-sampling***	***Over-sampling***	***Under-sampling***
content-length	1 : 0.3313	2 : 0.2208	1 : 0.2531
content-encoding gzip	2 : 0.2070	1 : 0.2512	2 : 0.2528
transfer-encoding chunked	3 : 0.0808	4 : 0.0862	3 : 0.0930
content-type text/html	4 : 0.0746	6 : 0.0388	8 : 0.0272
vary accept	5 : 0.0487	3 : 0.0904	4 : 0.0694
server apache	6 : 0.0408	7 : 0.0375	5 : 0.0400
cache-control max-age	7 :0.0263	5 : 0.0487	6 : 0.0386
connection keep-alive	8 : 0.0250	8 : 0.0280	7 : 0.0383
cache-control no-store	9 :0.0219	12 :0.0187	11 : 0.0204
pragma no-cache	10 : 0.0213	10 : 0.0213	9 : 0.0271
server nginx	11 : 0.0202	9 :0.0226	10 :0.0207
cache-control private	12 : 0.0136	15 : 0.0170	13 : 0.0150
expect-ct max-age	13 : 0.0135	14 : 0.0171	17 : 0.0104
x-content-type-options nosniff	14 : 0.0132	19 : 0.0099	22 : 0.0048
connection close	15 : 0.0129	20 : 0.0093	18 : 0.0093
cache-control must-revalidate	16 : 0.0122	16 : 0.0118	16 : 0.0118
via 1.1	17 : 0.0094	11 : 0.0189	14 : 0.0138
vary age	18 : 0.0089	18 : 0.0102	12 : 0.0150
cache-control no-cache	19 : 0.0074	17 : 0.0102	19 : 0.0091
strict-transport-security max-age	20 : 0.0052	13 : 0.0189	20 : 0.0090
x-xss-protection	21 : 0.0041	22 : 0.0059	15 : 0.0121
cache-control public	22 : 0.0017	21 : 0.0072	21 : 0.0089

Table 6-2.
The Top 3 HTTP Header Features from Prior Research Were Consistent
in Sampling Scenarios

11 HTTP Header Features from Prior Research Ranked			
Feature	***No-sampling***	***Over-sampling***	***Under-sampling***
content-length	1 : 0.4473	1 : 0.3585	1 : 0.3753
content-encoding gzip	2 : 0.2277	2 : 0.3077	2 : 0.3481
content-type text/html	3 : 0.1653	3 : 0.0999	3 : 0.0949
server apache	4 : 0.0485	5 : 0.0561	4 : 0.0522
cache-control max-age	5 : 0.0242	4 : 0.0621	5 : 0.0375
server nginx	6 : 0.0236	6 : 0.0368	6 : 0.0245
cache-control no-cache	7 : 0.0183	8 : 0.0184	8 : 0.0150
cache-control private	8 : 0.0148	7 : 0.0213	7 : 0.0150
cache-control no-store	9 : 0.0135	9 : 0.0142	9 : 0.0134
cache-control must-revalidate	10 : 0.0085	11 : 0.0118	10 : 0.0121
cache-control public	11 : 0.0083)	10 : 0.0132	11 : 0.0120

6.6.1.1 Features Identified in Previous Works

Researchers [40], [49], [75], [79] used `content-length` and `content-encoding` headers in their research on malicious websites and behavior, with `content-length` being a measure of the length in bytes of the content of the HTTP `request` or `response`. The `content-length` header is especially descriptive because there can be

great variation in this feature and there is no standard maximum `content-length` for `responses`. `Content-encoding` specifies the compression scheme used for the content of the HTTP `requests`. `Gzip`, `compress`, `deflate`, `identity`, and `br` are all different types of encodings, but the `identity` value indicates no compression of the content. Compression or zipping is a well-known technique for preventing security scanners from flagging on signatures in the content. Security scanners will raise an alert if incoming or outgoing content matches on a known malicious pattern, also referred to as a signature. We observed that the `gzip` encoding is of particular importance as noted in Table 6-1 and is studied with other zipped encodings by [40]. We were not surprised by its inclusion in the list of HTTP header features that are important for malicious website detection. The `content-length` header and the `gzip` value of `content-encoding` are ranked highly in all three cases, further validating their importance and inclusion in prior and future work. Tao et al. [77] used the `content-type` header in their HTTP feature set gathered over a session.

Further review of this list showed a large number of `cache-control` directives (six of 22) present in our list. The `cache-control` header specifies details about the caching mechanisms and can be present in HTTP `requests` and `responses`. In total, the six `cache-control` directives (`max-age`, `no-store`, `no-cache`, `must-revalidate`, and `public`) hovered around the middle of our rankings, with `max-age` being ranked as high as 5th in the under-sampling case and `public` and as low as 22nd in all three cases. In our literature review, we found that [212] examined the `cache-control` header, though it was not heavily used elsewhere. Since six of the top 22 features were related to the `cache-control` header, we validated the need to collect the

cache-control header and further found that six specific directives can be used to detect malicious websites and should be included in further studies.

In addition, we found that features specified in the server header can help detect malicious websites. In practice, the use of the server header is not recommended, since it could leak information about the website to the benefit of attackers. The inclusion of the server header in a response does not necessarily show a positive correlation to a website being malicious, but our work showed that the server header should be collected and that certain details about the server (whether it is an apache and nginx server) can help detect malicious websites. This validates the inclusion of the server header used in previous work [49] as a detector of malicious websites.

6.6.1.2 New Features Identified

The transfer-encoding header with the value chunked is viewed as a simple way to evade security scanners and its presence on this list of HTTP header features for malicious website identification is justified. This feature has not been studied in previous works, but is ranked highly in all sampling scenarios. The specific value of chunked indicates that content will arrive in chunks, thus making it harder to signature. To build intuition, consider the challenge of a security scanner that must piece together various chunks of data in order to make a determination on whether or not the content is malicious or hits a security signature. Having chunked data can make this problem more challenging.

The vary header, including a value of accept and the value of the age directive, are on our list, the former being highly ranked in all three scenarios with rankings of 5, 3, and 2, respectively, for no-sampling, under-sampling, and over-sampling. The vary

header describes behavior of the HTTP cache and tells the HTTP cache on the client which fields should be extracted from the `response` versus those that can originate from the HTTP cache. In our experiment, we found that `vary` specified the `accept` and `age` headers in our top 22, though the value of `accept` was consistently highly ranked. This header is also somewhat unique because its fields specify additional headers that should be processed differently by the client.

The `via` header with a value of `1.1` was also flagged for further investigation, though it did not appear in previous works and was not necessarily mapped to a known threat. The `via` header describes proxy behavior in several ways. In our experiment, the `1.1` indicated the protocol version of HTTP. Although proxies are known to be used in malicious activity, the evidence from this experiment was not strong enough to conclude that this was the case. Nevertheless, the `via` header, if equal to `1.1`, should be collected during future work with a focus on the protocol version, despite the fact that these values are not associated with a well-known threat or technique.

The inclusion of the `pragma` header in our list was of particular interest. Its value of `no-cache` did not rank very high, but it is a general header for HTTP/1.0 (not the current version) and its behavior, when present in `response`s, is not defined in the HTTP specification. To our knowledge, this header and its respective values are not associated with any known threat, but we recommend its use and further exploration since it represents an unpredictable part of the HTTP specification (undefined behavior when included in `response` headers) and was on our list of 22 features.

The `keep-alive` and `close` values for the `connection` header indicate whether or not the connection is to be kept open or closed and are not linked to any

known threats that we have identified. However, their importance is noted and they should be studied further.

The `expect-ct` header is another header used for defensive purposes, and `max-age` is a specific directive for this header. This header specifies that the browser checks the website's certificate to ensure that it is listed in the public Certificate Transparency logs. This header is set by the server requested. Because of the appearance of this header in our list, we recommend that it be examined and that the `max-age` directive be included in future feature sets. The presence of the `strict-transport-security` header informs the browser that the website should only be accessed over HTTPs and not over HTTP. The presence of the `x-xss-protection` header tells the browser to stop loading the page if the browser detects a cross-site scripting attack. The `x-content-type` header with value of `nosniff` tells the browser not to attempt to interpret the multipurpose internet male extension (MIME) type sent. Older browsers would attempt "MIME sniffing," where the browser would attempt to interpret the content and execute/render the contents. Doing so enables attackers to lie about the content type as a mechanism for hiding malicious code and objects. With the `nosniff` value in the `x-content-type` header, attackers cannot lie about the content type because the browser will not render or execute a content type if it detects a different type than the type specified.

6.6.1.3 Features Ranking Analysis

We observed that the top two features, both of which were prevalent in prior research, had an importance much higher than the remaining 20 features (`content-length` had an importance of 0.33 and `content-encoding gzip` had an importance

of 0.21 without sampling) both with and without sampling. We also observed that the feature rank and importance were similar when considering over-sampling or under-sampling. We also observed that the top eight features were the same with and without sampling. The cumulative importance of these eight features was 0.83, 0.80 and 0.81 for no-sampling, over-sampling and under-sampling, respectively.

Table 6-2 provides the feature rank and importance for the 11 features gathered from prior research. Compared to the 22 features, the first two features had higher importance (0.45 instead of 0.33 and 0.23 instead of 0.21) in the case of no-sampling. The combined feature importance for the top two features ranged from 0.66 to 0.72 for no-sampling, over-sampling, and under-sampling. As for the 22 features on our list, the feature rank and importance were similar when considering over-sampling or under-sampling. We also observed that the top five features were the same with and without sampling. The overall importance of these five features was 0.91, 0.88, and 0.91 for no-sampling, over-sampling, and under-sampling, respectively.

Overall, we identified 11 features that, to the best of our knowledge, have not been used for malicious website detection. The other 11 we identified have been used by prior researchers. The new features accounted for roughly a third of overall feature importance (32%, 31.55%, and 30.24% in the no-sampling, over-sampling, and under-sampling scenarios, respectively).

6.6.2 RQ2: Do the Additional Features Identified Improve Malicious Website Detection?

We went on to investigate model performance for the test dataset when using the no-sampling, under-sampling, and over-sampling. We compared the results of using our

expanded feature set of 22 features to the results of using the 11 features previously identified from prior research. Tables 6-3 and 6-4 provide the FPRs, FNRs, ACCs, AUCs, Precs, Recs, and MCCs. Tables 6-5 and 6-6 show these metrics for the feature transformation cases. We focused on the MCC to drive the discussion regarding our results. The best result in each column is bolded. A "/" separates the metric for models built with the 11 features from that of the models built with the 22 features.

Table 6-3.
Identified HTTP Header Features Outperformed Features from Prior Research inn Sampling Scenarios

Model Performance (11 HTTP Header Features from Prior Research / 22 Identified HTTP Header Features) in Sampling Scenarios															
Model	***No-sampling***					***Over-sampling***					***Under-sampling***				
	FPR	***FNR***	***ACC***	***AUC***	***MCC***	***FPR***	***FNR***	***ACC***	***AUC***	***MCC***	***FPR***	***FNR***	***ACC***	***AUC***	***MCC***
KNN	**0.0057**/ 0.0061	0.7865/ 0.6932	0.8929/ 0.9047	0.6038/ 0.6502	0.3923/ 0.4865	**0.0268**/ 0.0975	0.7512/ 0.1252	0.8791/ **0.8988**	0.6109/ 0.8885	0.3254/ **0.6548**	**0.0321**/ 0.1083	0.7773/ 0.1251	0.8711/ **0.8894**	0.5952/ 0.8832	0.2762/ 0.6347
LR	0.0774/ 0.0840	0.4537/ 0.3302	0.8737/ 0.8839	0.7343/ 0.7928	0.4563/ 0.5367	0.1795/ 0.1644	0.1621/ 0.1058	0.8227/ 0.8431	0.8291/ 0.8648	0.5012/ 0.5595	0.1789/ 0.1647	0.1596/ 0.1126	0.8235/ 0.8420	0.8307/ 0.8613	0.5037/ 0.5546
RF	0.0965/ 0.0819	0.1336/ 0.1470	0.8986/ 0.9096	0.8848/ 0.8855	0.6510/ 0.6714	0.1166/ 0.1166	0.1033/ 0.0882	0.8850/ 0.8870	0.8900/ 0.8975	0.6349/ 0.6451	0.1290/ 0.1319	0.0966/ 0.0815	0.8751/ 0.8746	0.8871/ 0.8932	0.6187/ 0.6243
AB	0.0828/ 0.0808	0.3226/ 0.2638	0.8860/ 0.8954	0.7972/ 0.8276	0.5449/ 0.5920	0.1787/ 0.1552	0.1176/ 0.1151	0.8291/ 0.8499	0.8517/ 0.8648	0.5324/ 0.5662	0.1716/ 0.1593	0.1462/ 0.1151	0.8316/ 0.8463	0.8410/ 0.8627	0.5224/ 0.5604
GB	0.0793/ 0.0794	0.2823/ 0.1983	0.8943/ 0.9051	0.8191/ 0.8611	0.5819/ 0.6414	0.1369/ 0.1225	0.1016/ 0.1042	0.8676/ 0.8798	0.8806/ 0.8866	0.6027/ 0.6243	0.1439/ 0.1284	0.0957/ 0.1025	0.8622/ 0.8749	0.8801/ 0.8845	0.5959/ 0.6157
ET	0.0983/ 0.0835	0.1277/ 0.1436	0.8978/ 0.9086	0.8869/ 0.8863	0.6516/ 0.6703	0.1188/ 0.1181	0.1025/ **0.0848**	0.8832/ 0.8861	0.8892/ **0.8984**	0.6316/ 0.6448	0.1309/ 0.1336	0.0983/ **0.0739**	0.8733/ 0.8740	0.8853/ **0.8961**	0.6145/ 0.6265
BC	0.0953/ 0.0811	**0.1235**/ 0.1445	0.9010/ **0.9105**	**0.8905**/ 0.8871	0.6603/ **0.6748**	0.1192/ 0.1170	0.1016/ 0.0882	0.8830/ 0.8867	0.8895/ 0.8973	0.6315/ 0.6445	0.1280/ 0.1424	0.1016/ 0.0789	0.8753/ 0.8657	0.8851/ 0.8892	0.6169/ 0.6095
NN	0.0972/ 0.0840	0.2067/ 0.1756	0.8885/ 0.9040	**0.8905**/ 0.8871	0.5992/ 0.6473	0.1227/ 0.1218	0.1100/ 0.0865	0.8788/ 0.8826	0.8895/ 0.8973	0.6199/ 0.6370	0.1221/ 0.1221	0.1151/ 0.0882	0.8787/ 0.8822	0.8851/ 0.8892	0.6175/ **0.6357**

Table 6-4.
Identified HTTP header Features Outperformed Features
from Prior Research in Sampling Scenarios (cont.)

Model Performance (11 HTTP Header Features from Prior Research / 22 Identified HTTP Header Features) in Sampling Scenarios						
	No-sampling		***Over-sampling***		***Under-sampling***	
Model	***Prec***	***Rec***	***Prec***	***Rec***	***Prec***	***Rec***
KNN	0.8466/ **0.8816**	0.2134/ 0.3067	**0.5803**/ 0.5719	0.2487/ 0.8747	0.5076/ 0.5461	0.2226/ 0.8747
LR	0.5126/ 0.5429	0.5462/ 0.6697	0.4102/ 0.4476	0.8378/ 0.8941	0.4118/ 0.4453	0.8403/ 0.8873
RF	0.5721/ 0.6081	0.8663/ 0.8529	0.5340/ 0.5381	0.8966/ 0.9117	0.5106/ 0.5093	0.9033/ 0.9184
AB	0.5494/ 0.5759	0.6773/ 0.7361	0.4238/ 0.4594	0.8823/ 0.8848	0.4258/ 0.4529	0.8537/ 0.8848
GB	0.5743/ 0.6007	0.7176/ 0.8016	0.4944/ 0.5215	0.8983/ 0.8957	0.4835/ 0.5102	0.9042/ 0.8974
ET	0.5693/ 0.6043	0.8722/ 0.8563	0.5294/ 0.5359	0.8974/ **0.9151**	0.5066/ 0.5080	0.9016/ **0.9260**
BC	0.5781/ 0.6110	0**.8764**/ 0.8554	0.5289/ 0.5373	0.8983/ 0.9117	0.5112/ 0.4908	0.8983/ 0.9210
NN	0.5488/ 0.5938	0.7932/ 0.8243	0.5193/ 0.5276	0.8899/ 0.9134	0.5192/ **0.5266**	0.8848/ 0.9117

Without sampling, the MCC was higher for all eight models when considering the 22 features instead of the 11 previously studied features (on average, 0.615 compared to 0.57). When over-sampling, the average MCC increased from 0.56 to 0.62 when considering the set of 22 features instead of the set of 11 previously studied features. With over-sampling, the MCC was higher for all eight models when considering the set of 22 features instead of the set of 11 previously studied features. When under-sampling, the average MCC increased from 0.545 to 0.61 when considering 22 features instead of the previously studied 11 features. With under-sampling, the MCC was higher for all eight models other than BC when considering the 22 features instead of the 11 previously studied features.

We performed feature transformation on the 22 features to determine whether there were combinations of features that improved performance. We used the feature transformation process in Section 3.4.3 to transformation the original 22 features were

transformed into 946 features. We then performed feature elimination on the 946 features using four different techniques from Section 3.4.3 and kept features selected by at least three of these techniques, leaving 36 in total.

We also determined whether PCA could reduce the 946 transformed features to some “n” number components while capturing the maximum variance. Using a cumulative scree plot, we found that by using 117 components, we were able to capture 95% of the variance for the 22 features and that by using 56 components, we captured 95% of the variance. Using these 117 and 56 components, we attempted to see how our models performed. Results are shown in Tables 6-5 and 6-6.

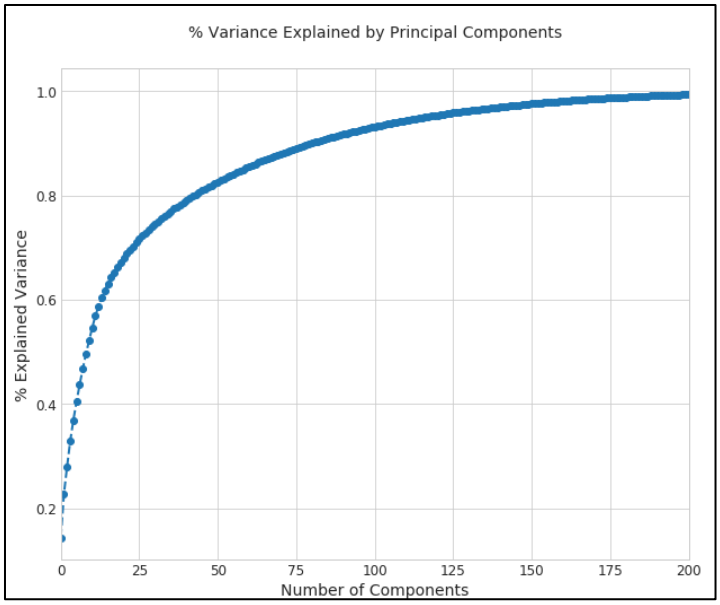


Fig. 6-1. 22 header features yielded 117 components

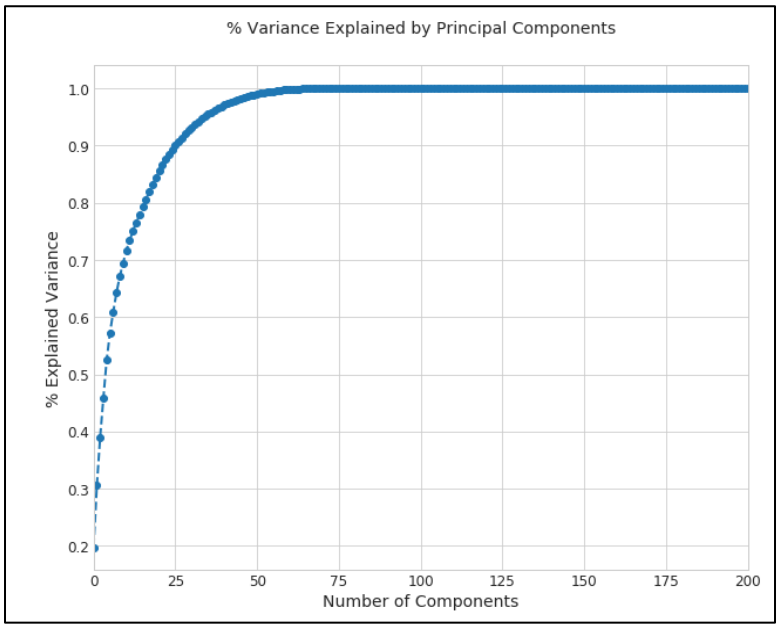


Fig. 6-2. 11 header features yielded 56 components

Table 6-5.
Identified HTTP Header Features Outperformed Features from Prior Research in Feature Transformation Scenarios

Model Performance (11 HTTP Header Features from Prior Research / 22 Identified HTTP Header Features) with Feature Transformation										
Model	***FT w/FS***					***FT w/PCA***				
	FPR	***FNR***	***ACC***	***AUC***	***MCC***	***FPR***	***FNR***	***ACC***	***AUC***	***MCC***
KNN	0.0806/ 0.0912	0.3705/ 0.1907	0.8817/ 0.8958	0.7743/ 0.8590	0.5137/ 0.6221	**0.0238**/ 0.0815	0.6638/ 0.1521	0.8931/ **0.9092**	0.6561/ 0.8831	0.4277/ 0.6688
LR	0.0870/ 0.0776	0.4058/ 0.4176	0.8715/ 0.8782	0.7535/ 0.7523	0.4734/ 0.4842	0.0828/ 0.0845	0.4067/ 0.2176	0.8751/ 0.8981	0.7552/ 0.8488	0.4816/ 0.6169
RF	0.0874/ 0.0909	0.2831/ 0.1899	0.8871/ 0.8962	0.8146/ 0.8595	0.5641/ 0.6232	0.0967/ 0.0830	**0.1285**/ 0.1462	0.8991/ 0.9087	0.8873/ 0.8853	0.6542/ 0.6696
AB	0.0858/ **0.0031**	0.4117/ 0.8815	0.8718/ 0.8829	0.7512/ 0.5576	0.4715/ 0.2907	0.0942/ 0.0804	0.1815/ 0.2075	0.8944/ 0.9030	0.8621/ 0.8560	0.6226/ 0.6327
GB	0.0825/ 0.0908	0.3142/ 0.2159	0.8873/ 0.8929	0.8015/ 0.8466	0.5516/ 0.6052	0.0923/ 0.0803	0.1638/ 0.1689	0.8983/ 0.9081	0.8719/ 0.8753	0.6385/ 0.6599
ET	0.0873/ 0.0910	0.2823/ **0.1890**	0.8873/ 0.8962	0.8151/ **0.8599**	0.5650/ 0.6236	0.0982/ 0.0829	0.1310/ 0.1462	0.8975/ 0.9088	0.8853/ 0.8854	0.6496/ 0.6699
BC	0.0856/ 0.0907	0.2848/ 0.1899	0.8884/ **0.8964**	0.8147/ 0.8596	0.5665/ **0.6237**	0.0974/ 0.0820	0.1294/ 0.1487	0.8983/ **0.9092**	0.8865/ 0.8846	0.6521/ **0.6700**
NN	0.0836/ 0.0907	0.2949/ 0.2268	0.8889/ 0.8916	0.8147/ 0.8596	0.5634/ 0.5978	0.0819/ 0.0820	0.2840/ 0.1789	0.8918/ 0.9053	0.8865/ 0.8846	0.5751/ 0.6492

Table 6-6.
Identified HTTP Header Features Outperformed Features from Prior Research in Feature Transformation Scenarios (cont.)

Model Performance (11 HTTP Header Features from Prior Research / 22 Identified HTTP Header Features) with Feature Transformation				
	FT w/FS		***FT w/PCA***	
Model	***Prec***	***Rec***	***Prec***	***Rec***
KNN	0.5376/ 0.5694	0.6294/ 0.8092	**0.6779**/ 0.6078	0.3361/ 0.8478
LR	0.5042/ 0.5277	0.5941/ 0.5823	0.5164/ 0.5797	0.5932/ 0.7823
RF	0.5499/ 0.5704	0.7168/ 0.8100	0.5732/ 0.6051	**0.8714**/ 0.8537
AC	0.5054/ 0.8493	0.5882/ 0.1184	0.5643/ 0.5949	0.8184/ 0.7924
GB	0.5532/ 0.5627	0.6857/ 0.7840	0.5744/ 0.6067	0.8361/ 0.8310
ET	0.5506/ 0.5703	0.7176/ **0.8109**	0.5687/ 0.6054	0.8689/ 0.8537
BC	0.5543/ **0.5710**	0.7151/ 0.8100	0.5711/ 0.6073	0.8705/ 0.8512
NN	0.5567/ 0.5596	0.7050/ 0.7731	0.5657/ 0.5986	0.7159/ 0.8210

For feature transformation with feature selection, the MCC was higher for the models (other than AB), when considering the set of 22 features instead of the set of 11 previously studied features. When considering the 22 features instead of the 11

previously studied features, the average MCC improved from 0.53 to 0.56. However, when considering the 22 features, FT w/FS reduced the average MCC from 0.615 to 0.56 when compared to no feature transformation.

For FT w/PCA, the MCC was higher for all eight models when considering the 22 features instead of 11 previously studied features. The MCC average also improved from 0.59 to 0.65. When compared to no feature transformation, the average MCC improved from 0.615 to 0.65.

When considering the effect of feature transformation on our model performance, we found that FT w/ FS worsened the average MCC, while FT w/PCA improved the average MCC. Thus, we demonstrated that, compared to the case without sampling and without feature transformation, feature transformation with PCA improved the results but feature transformation with feature selection worsened them. We observed improvement in 38 of the 40 models when adding the new features and postulated that additional HTTP header features can improve malicious website detection.

6.6.3 RQ3: Do our Results Change with No-sampling, Under-sampling, and Over-sampling Scenarios?

In posing RQ3, we addressed the sensitivity of our approach and its robustness in dataset sampling. Sampling is important in malicious website classification because researchers, ourselves included, work with various datasets that may or may not have class imbalance. In other words, there may be more malicious than non-malicious websites used in the training and test sets or vice versa. Currently, no standard exists for whether or not to perform sampling, nor is there a set standard regarding how much of a class imbalance between malicious and non-malicious websites should be present for

training and testing malicious website detection models. Hence, exploring whether or not sampling had an effect was worthwhile. We compared two parts of our results – the feature rankings and the overall performance of our classifiers.

The features rankings were stable, with the top eight from Table 6-2 and the top five from Table 6-3 being the same. The MCC was 0.5671, 0.5599, and 0.5457 for the 11 features in the no-, over-, and under-sampling cases, and was 0.6150, 0.6220, and 0.6076 for the 22 features in the no-, over-, and under-sampling cases. Thus, we observed that result, feature ranking, and importance, were fairly consistent in the different sampling scenarios.

6.6.4 RQ4: Does Hyperparameter Tuning and Cross-Validation Improve our Results?

In this last step, we used RQ4 to investigate how we could add additional assurance to our approach and evaluated the effect of tuning our models. After researching common techniques, we decided to perform hyperparameter tuning and cross-validation on our dataset and to re-evaluate our models with a 70:30 split of train to test data instead of the initial 80:20 split. By doing so, we could investigate that our models were not overfit, we could potentially improve our models, and we could ensure that our observations were not dependent on the initial 80:20 split of data.

We performed hyperparameter tuning and cross-validation on the best performing models in each of the five scenarios – no-sampling, over-sampling, under-sampling, feature transformation with feature selection, and feature transformation with PCA. In the 80:20 case, all five models improved, but the average MCC only increased from 0.652 to 0.657, suggesting validity of the results in Tables 6-3, 6-4, 6-5, and 6-6.

In the 70:30 case, tuning and cross-validation improved three of five models for the 70:30 split, but the average MCC only increased from 0.653 to 0.655. Without tuning and cross-validation, the average MCC was 0.653 and 0.652, respectively, with the 70:30 and 80:20 splits. With tuning and cross-validation, the average MCC was 0.655 and 0.657, respectively, with the 70:30 and 80:20 splits. The small difference between results in the different splits suggests that we were not dependent on the train/test split. Results are shown in Table 6-7.

Table 6-7.
Cross-Validation and Hyperparameter Tuning Slightly Improved HTTP Header Models

Cross-Validation and Hyperparameter Tuning HTTP Models			
Model	***Scenario - Split***	***MCC***	***Scoring Metric***
ET	No-sampling - 70:30	0.6497	balanced accuracy
KNN	Over-sampling - 70:30	**0.6784**	precision weighted
RF	Under-sampling - 70:30	0.6414	precision weighted
RF	FT w/ FS - 70:30	0.6301	precision micro
BC	FT w/PCA - 70:30	0.6773	Recall
BC	No-sampling - 80:20	0.6764	f1 weighted
KNN	Over-sampling - 80:20	0.6722	precision weighted
NN	Under-sampling - 80:20	0.6400	Recall
BC	FT w/FS - 80:20	0.6249	f1 weighted
BC	FT w/PCA - 80:20	0.6713	Recall

In this fourth research question, we determined two ways of validating our results. First, we performed hyperparameter tuning and cross-validation. Secondly, we rebuilt and performed hyperparameter tuning and cross-validation on our eight models on a 70:30 split of the data. We observed that while hyperparameter tuning and cross-validation improved our results, the improvements were not large.

6.7 Conclusion

This chapter detailed our comprehensive evaluation of HTTP header features to assess whether additional HTTP header features could improve malicious website detection. We analyzed HTTP headers from 6,021 malicious websites and from 39,853

benign websites. We used a dataset of malicious websites identified by Cisco Talos and used a set of benign websites from the Alexa Top 1M (Dataset 1). We collected 672 HTTP header features from these websites, identifying 22 for further analysis, including 11 that were newly identified. We applied eight models, ensuring the robustness of our methodology by performing no-sampling, over-sampling and under-sampling.

Of the 22 features studied, we found eight to be consistently ranked as the most important features, representing 80% of feature importance. Of those eight important features, three were features identified by our approach. The average MCCs for the selected 22 features were consistently better than for the 11 previously studied features. When considering the 22 selected features, FT w/PCA increased the MCC. Our results indicated the existence of a broader set of HTTP header features that are applicable for malicious website detection, beyond those that have been commonly studied by prior scholars. In addition, our results showed consistency over various scenarios.

Chapter 7: Combined Web Request Features Analysis

7.1 Introduction

In this chapter, we present a comprehensive evaluation of discovery of features for malicious website detection with webpage content, URL, and HTTP header features instead of *a priori* selection of features. We do so by collecting features from a `response` to a web `request`. Our dataset (Dataset 1) consists of benign websites from the Alexa Top Domains [112] provided by [176]. The malicious websites consist of phishing webpages, drive-by downloads, and other malicious websites including command and control (C2) URLs provided by the Cisco Talos Intelligence Group [177]. We apply a series of feature selection techniques to discover features suitable for detection of malicious websites. We investigate their detection performance using unsupervised and supervised learning algorithms in various sampling and feature transformation scenarios. We compare the detection performance of the discovered features to the detection performance provided by features from prior research. Overall, we find:

- The discovery approach identifies features used in prior research, and new features and feature combinations;
- The discovery approach produces features that yield similar (and slightly better on average without model tuning and slightly worse with model tuning) performance to features from previously published but requires fewer features for the same level of performance; and

- The discovery approach identifies features that produce better meta-features via feature transformation further demonstrating benefits over selecting features *a priori*.

We make the following contributions:

- We demonstrated the potential for discovering features for malicious website detection by achieving a best-classifier ACC, AUC, MCC, Prec, and Rec of 98.38%, 0.9464, 0.9174, 0.9555, 0.8982, respectively, with tuning and overall averages of 96.62%, 0.9251, 0.8432, 0.8560, and 0.8723, respectively, across several machine learning models built with default parameters; and
- We showed that new features must be discovered and evaluated for their ability to detect malicious websites by demonstrating that supervised models built from discovered features, 12 of which were newly identified and 22 of which had been used in prior research, outperformed models built with features from prior research by an average MCC of 0.0208 with 66% fewer features when using default parameters.

7.2 Related Research

Machine learning has been applied in many cybersecurity studies and has shown potential to detect various threats and malicious websites. Three threats are commonly detected in prior research – phishing webpages, drive-by downloads, and C2 infrastructure.

Ma [35] used the URL structure and host-based properties gathered from other sources (IP denylists, WHOIS, domain name properties, and geographic properties) with naïve Bayes [96], and LR [92] classifiers. They continued in [36]-[37], using similar

features and adding the use of online learning [217] to detect malicious URLs. CANTINA [24] relied primarily on features from the webpage content and created heuristics, evaluating the framework based upon 100 legitimate URLs from a prior study [123] and 100 URLs from PhishTank [113]. CANTINA+ [46] improved upon this approach by adding features and training and by testing models built from various machine learning methods. The authors relied on subject matter expertise to select features and used search engine features derived from search results. Whittaker et al.[42] used an LR classifier and used millions of URLs for evaluation. Marchal et al. [91] took an approach similar to that of the CANTINA+ authors, gathering 212 features, differentiating between languages on the webpage, and analyzing the URL structure more than CANTINA and CANTINA+ authors had. Their method used a GB [94] algorithm. Phishmon [83] leveraged HTTP header features for their phishing detection mechanism and Li et al. [218] used feature transformation to perform better phishing detection.

JSAND creators Cova et al. [65] identified malicious JavaScript with 10 features associated with drive-by downloads. Their approach relied on instrumenting a browser, executing the code, and recording the events. Rieck et al. [67] used Cujo to perform static and dynamic analysis of JavaScript. The static analysis relied on lexical tokens and the dynamic analysis relied on known attack patterns. The sequences from static and dynamic analysis were transformed into Q-grams – a sequence of "q" words within the code execution – that were then used to train an SVM. Curtsinger et al. [68] used Zozzle to perform static analysis by first de-obfuscating the JavaScript and creating features from two parts – the context in which it appeared (try/catch block, etc.), and some text. They used contexts relevant to malicious JavaScript detection. Features were selected and

used to train a naïve Bayesian classifier on 919 malicious entries and 7,976 benign samples and achieved a false positive rate of 0.0003% on 1.2 million files. Revolver was used to examine the AST created from the JavaScript, to create sequences of nodes, and to compare the similarity to known malicious sequences. Researchers [45] used Prophiler to detect drive-by downloads with features commonly used in phishing detection (webpage content). They trained their model on 787 samples of drive-by downloads with HTML elements, static JavaScript features, URL features, and features from DNS. Zhang et al. [78] used Arrow to detect drive-by downloads with HTTP traces pulled from logs instead of from the JavaScript. JavaScript analysis was the source of features used in studies to detect drive-by downloads, though researchers also have used other features commonly associated with phishing detection.

Authors in [75]-[76] performed clustering on high-level features (total number of HTTP `request`s, number of `GET`/`POST` `request`s, `response` lengths, etc.) and on lower-level features such as specific headers, and creating HTTP traffic clusters to derive signatures for C2 (bot) URLs. Researchers [80] used ExecScent which focused on clustering `request`s and built templates for detection from HTTP traffic clusters, and derived signatures from the URL path length, query component, user-agent string, and other headers. These features and similar features have been used in other studies as well, with [55] using them to compare distances of clusters of HTTP `request`s by extracting the URL path, URL parameters, and URL method (`GET`, `POST`, etc.). Zarras et al. [82] used header chains (sequences of HTTP headers) for their detection method, creating signatures from clustering. Researchers [219] also aimed to detect bots and gathered "flow-based" features (extracted from network traffic) over a period of time to do so.

Yadav et al. [209] sought to detect DGAs [53] by examining features purely from domain names.

Although researchers aimed to detect differing threats, some were able to detect multiple threats with various types of features. For example, [47] used features to detect phishing webpages and drive-by downloads, while [49] selected various feature types – webpage content, flow-based features, URL features, etc. – and leveraged [220] as their dataset.

7.3 Research Questions 5–7

7.3.1 Research Question 5

Prior studies have relied on preconceived notions of relevant (*a priori*) features – URL length, `<iframe>`s, etc., for detection. This reliance has demonstrated success however attacks change over time [221], as do the technologies used by attackers and as well as developers of benign websites. There is a need to re-evaluate the features used to detect malicious websites. For example, HTML5 [222], released in 2014, introduces new tags (elements). Hence, features that were new to HTML5 could not have been included in research prior to 2014. Additionally, there are techniques such as feature selection [50] that can be employed to discover features which may be more applicable to the detection of malicious websites. RQ5, stated as follows:

RQ5: Is feature discovery feasible for malicious website detection?

7.3.2 Research Question 6

Even if feature discovery is feasible, there is no guarantee that it is better than selecting features *a priori*. To date, there is little insight into how discovered features' detection ability compares to those from prior research. RQ6, with which we investigated

how discovered features performed compared to those gathered from prior research, is stated as follows:

RQ6: How do discovered features' detection ability compare to those from prior research?

7.3.3 **Research Question 7**

Once we've established the feasibility of discovered features and compared their detection ability to the detection ability of *a priori* features from prior research, we then investigated operational constraints. A constraint within an operational scenario is that a network is exposed to various threats simultaneously. Denylists (among other tools) are used to prevent communication with the malicious website. Some prior researchers worked successfully to identify specific threats: [91] focused on phishing, [67] focused on drive-by downloads, and [75]-[76] focused on detecting C2 infrastructure. In this study, we gain insight into whether features could be used to detect a group of threats, regardless of their nature.

Success in an operational scenario also depends on the availability and choice of features used. For example, phishing is typically best detected from HTML on a webpage, drive-by downloads are best detected by the JavaScript, and C2 infrastructure is best detected by the URL structure or HTTP headers. When a user (or service) retrieves a malicious website, there may not be information about the type of threat or the relevant features for detecting the malicious website. Furthermore, a website could be a phishing website, could result in a drive-by download, and could serve as C2 infrastructure. Additionally, some features that are useful for detection may not be available in a timely manner. DNS information, search engine results, and WHOIS features, for example, all

show promise for detection, but we cannot guarantee that these features are available when determining whether to block communication with a website. This chapter leveraged features derived from the web `response`, which was derived directly from the website.

It is unclear whether discovering features can be applicable to an operational environment. Hence, we arrived at RQ7, stated as follows:

RQ7: Can a discovery approach be applied to several threats when only features from a web response are available?

7.4 Methodology

Figure 7-1 provides an overview of the seven-step analytical process we used in this chapter which can be viewed as a culmination of Chapters 4-6. Images courtesy of Pixabay [22].

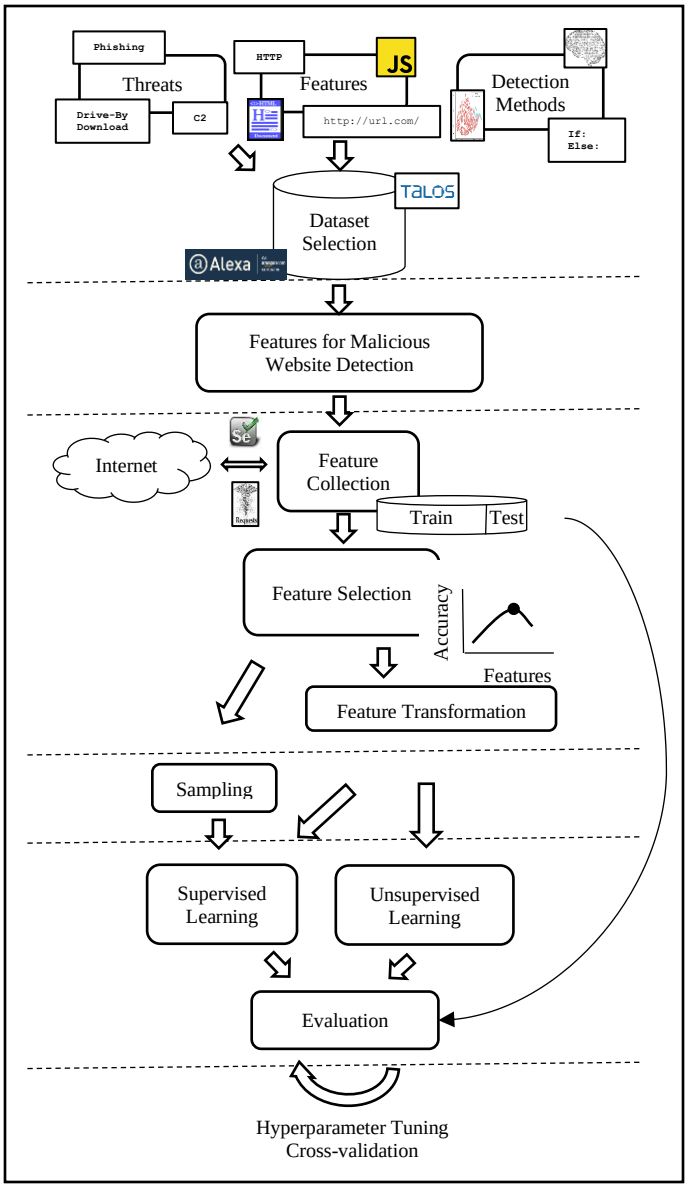


Fig. 7-1 A process for discovery and evaluation of features for malicious website detection

7.4.1 **Dataset Selection**

Our dataset (Dataset 1) consists of two portions – benign entries from the 39,877 Alexa top domains, and 6,894 malicious entries provided by Cisco Talos Intelligence Group [15]. The Alexa top domains have been used in several studies including [43]. The malicious entries are a mix of websites and include several threats (phishing, drive-by

download, and C2). Both datasets are provided by external organizations and are not hand-selected or created for the purpose of this study.

Because we obtained the dataset of malicious websites from a third party, we exerted no influence over its size. To select a size for the benign dataset, we first surveyed prior research. Dataset sizes varied widely in prior scholarship. Researchers [68] used 919 malicious entries and 7,976 benign samples in their training and they evaluated their method on 1.2 million files. On the other hand, authors [91] used 1,213 malicious samples and 5,000 benign samples in their training and used 1,553 entries in their testing set. For this research, we made the decision to select 39,877 entries for our benign dataset. Both small and large datasets have their own respective advantages, with smaller datasets allowing for focus and deeper analysis of a few samples, and larger datasets potentially being more representative. In order to account for the dataset imbalance, we performed sampling which is discussed in Section 7.4.4.

7.4.2 Features for Malicious Website Detection

This chapter leveraged features from Chapters 4-6. Please refer to Chapters 4-6 Sections 4.4, 5.4, and 6.4 for complete details.

7.4.3 Feature Collection, Selection, and Transformation

7.4.3.1 Feature Collection

We performed feature collection in August of 2018, recording the content and the HTTP headers from `GET requests` and discarding entries for which the `GET requests` failed. We retrieved 34,742 entries from among the top 39,877 domain names in the Alexa Top 1M and retrieved 4,441 entries from among the 6,894 Cisco Talos entries. Failed `requests` were due to causes including: timeouts, firewalls blocking our IP

address, or the web URL no longer being valid (more prevalent in the malicious dataset). We collected 46,580 features in total – 28,162 derived from the URL, 17,746 derived from the webpage content, and 672 derived from the HTTP `response` headers.

7.4.3.2 Feature Selection

We split our dataset into two portions – a training set (80% of total data), and a testing set (20% of total data). The training set is used for feature selection and model building, and the testing set is placed aside for evaluation. To perform feature selection we used the feature selection process with XgBoost described in Section 3.3.2 and the initial steps yielded a list of sets each containing three elements - a threshold 'thresh,' number of features 'n,' set of features 'f,' and an accuracy. Our goal was to achieve the best performance with few features. Thus, we iterated through the sets consisting of threshold, number of features, and accuracy (with 'n' decreasing) and looked for a relative maximum in accuracy. The presence of a relative maximum in accuracy, as 'n' decreases, is how we determined which set of features to use for detection. We find a relative maximum accuracy at n=36 features as shown below.

```
Thresh=0.001, n=105, Accuracy: 97.78%
                    …
Thresh=0.006, n=43, Accuracy: 97.72%
Thresh=0.007, n=36, Accuracy: 97.75%
Thresh=0.009, n=26, Accuracy: 97.69%
```

We calculated the correlation [223] of the features and observed that two features had high correlation to other features in the list. We removed the features with high correlation and arrived at 34 features. These 34 features are the *discovered* features. We have also identified 99 features used in prior research which are referred to as the 99 *a priori* features.

7.4.3.3 Feature Transformation

Although we have 34 *discovered* and 99 *a priori* features to build our detection models, we also compared their effectiveness by comparing the detection abilities of features they can produce. As such, we performed feature transformation on the 34 *discovered* and 99 *a priori* features to form additional features and evaluate their ability to detect malicious websites. We performed two types of feature transformation scenarios – feature transformation with feature selection (FT w/FS) and feature transformation with principal component analysis (FT w/PCA) from Section 3.4.3 in order to build a better understanding of how the *discovered* features detection ability compares to that of the *a priori*. Both involved first transforming the 34 *discovered* and 99 *a priori* features into new features which we referred to as the transformed features. Since feature transformation produced many features, we then performed feature selection with additional selection techniques and dimensionality reduction with PCA respectively to generate a smaller set of features (components in the case of PCA) to build our detection models.

Once we transformed the 34 and 99 features, we then performed feature selection on the transformed features using four different techniques: Correlation as used in [189], Select K Best (scoring function chi-square), Recursive Feature Elimination (RFE), and Select From Model in the feature transformation with feature selection case. The choice of these techniques is motivated by prior research and current data science techniques. We input transformed features created from the addition, multiplication, and division transformations into each of these four techniques which produced four sets of features. We kept (selected) the transformed features that at least three of these techniques select

as relevant. For the feature transformation with PCA case, we performed PCA on the transformed features to create ‘n’ components, mixtures, or combinations of variables that capture the maximum variance, which are then used to build detection models. By using a cumulative scree plot, we identified components that capture maximum variance between the features within the data.

7.4.4 **Sampling**

Our dataset has imbalance, 6,894 malicious websites to 39,877 benign websites, which could influence the detection performance of models built in this study. To account for this potential impact, we created three sampling scenarios for the training data from which we will build supervised learning models no-sampling – training data are used as is; over-sampling – malicious websites are over-sampled with the SMOTE technique [186] provided by [187] to equal the number of benign websites; and under-sampling – benign websites are randomly under-sampled to equal the number of malicious websites.

7.4.5 **Unsupervised and Supervised Learning**

We leveraged unsupervised and supervised learning to build malicious website detection models. We captured the ACC, AUC, MCC, Prec, and Rec for each model. We focused our discussion on the MCC.

Unsupervised learning in the form of clustering and anomaly detection is commonly used to detect malicious websites and traffic and has been used in prior research as in [65] and [75]-[76]. Clustering is more applicable to distinguishing and discovering different classes within the data. For example, [80] used clustering to distinguish HTTP traffic among different families of malware. Since we have only two data classes (malicious and non-malicious), we chose to leverage an anomaly detection

approach where benign websites are the normal case and the malicious websites are anomalous. We built unsupervised models with default parameters using Autoencoders [192] and a One Class SVM from the 34 discovered features, 99 *a priori* features, transformed features using the FT w/FS technique, and transformed features using the FT w/PCA technique.

We also leveraged nine supervised learning algorithms to detect malicious websites because supervised learning is more common: nearest neighbor [111], ensemble methods [182], and NNs [183]. The choice of building nine models was motivated by two factors. First, we wanted to explore performance of various models built with the *discovered* and *a priori* features to gain a more thorough understanding of the features' detection ability. Second, we found that [47] leveraged seven different supervised algorithms which were combined with a voting [191] algorithm. Of our nine models, five are ensemble methods and provide a measure of feature importance [185] based on the Gini Impurity: AB [101], ET [97], RF [98],GB, and XGB. Feature importance enabled us to examine which features contribute the most to the classification decision and allows us to create a ranking of the most importance features for detection. We also built a Voting classifier (V) [191] created from the RF, ET, and GB classifiers. The other three models did not provide a measure of feature importance however provided additional insight into how the selected features perform across well-known machine learning algorithms and enable a more thorough comparison of the two sets of features. They were the BC [184], an ensemble method, KNN, a nearest neighbor method, and NNs. When building the models, we chose to use the default parameters provided by [29] for the respective models in an effort to reduce subjectivity and to be consistent. Our analysis in Sections

7.5.2 through 7.5.4 was done on the models built with default parameters. We did however, perform hyperparameter tuning and cross-validation where we varied the model hyperparameters and adjusted the training and testing data with Kfold [190] validation. Doing so attempted to improve our models and provided better insight into their performance of the features by lessening the reliance on the initial 80:20 training to test split of data.

7.4.6 Hyperparameter Tuning and Cross-Validation

In our last step, we performed hyperparameter tuning and cross-validation on our supervised models to improve our models and validate our results built from default parameters. We chose the best performing model from each scenario – no-sampling, over-sampling, under-sampling, FT w/FS, and FT w/PCA and tuned the hyperparameters and performed cross-validation. We used StratifiedKFold [190] for 10-fold cross-validation [26] and explored several scoring metrics (accuracy, precision, recall) provided by [29] when performing Grid Search [224] in our attempt to maximize our performance.

For added assurance that our results were not a result of our initial 80:20 train to test split, we repeated our training, sampling, and feature transformation approach with a 70:30 split of training to testing data and again performed hyperparameter tuning and cross-validation. We also performed hyperparameter tuning and 5-fold cross-validation on the Voting classifier.

7.5 Results

7.5.1 Unsupervised Results

In the unsupervised case, we observed an average ACC, AUC, MCC, Prec, and Rec of 88.72%, 0.8076, 0.5491, 0.7277, and 0.8141, respectively, with discovered

features and observed an average ACC, AUC, MCC, Prec, and Rec of 88.39%, 0.7556, 0.4699, 0.6237, and 0.8264, respectively, with *a priori* features. Full results are shown in Figures 7-2 and 7-3.

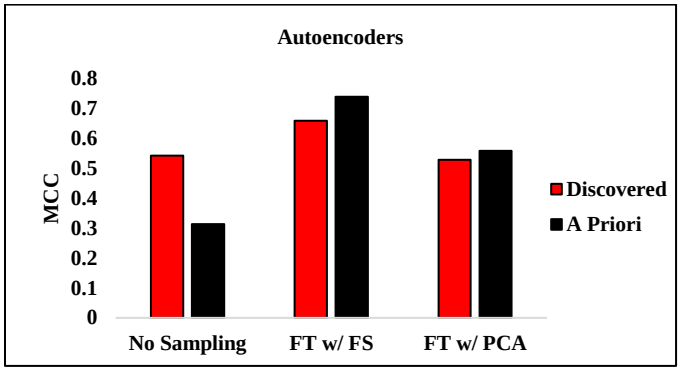


Fig. 7-2. Autoencoders perform better with transformed features

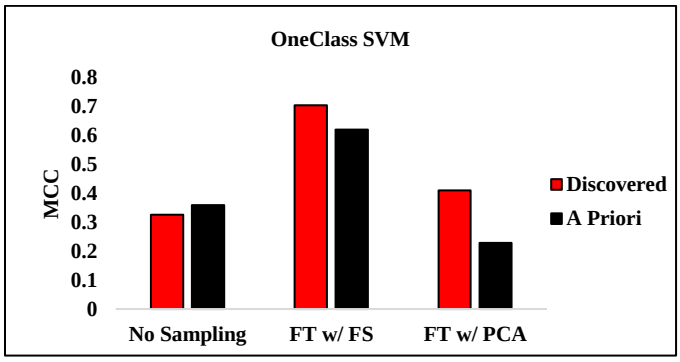


Fig. 7-3. One-class SVMs perform well with feature transformation with feature selection and perform poorly with feature transformation with PCA

The unsupervised models did not perform well, though we saw improvement (on average) when using the discovered features versus the *a priori* – an average increase in MCC of 0.0793. The detection results were not great; hence, we focused most of our analysis on supervised methods.

7.5.2 **Feature Selection Importance**

Rankings of the 34 discovered features are shown in Table 7-1 in the no-sampling, over-sampling, and under-sampling scenarios. Shaded rows mark the new features. Rank and importance are separated by a ":" character.

Table 7-1.
Feature Selection Identified 22 Features Used in Prior Research and 12 that Were New

34 Discovered Features Ranked			
Feature	***No-sampling***	***Over-sampling***	***Under-sampling***
Total HTML Tags	1 : 0.1311	1 : 0.1318	1 : 0.1138
URL Length	2 : 0.1285	2 : 0.0845	2 : 0.0926
Total Extensions in URL	3 : 0.1137	3 : 0.0717	4 : 0.0622
Count of 'w' character	4 : 0.0734	5 : 0.0575	5 : 0.0563
Count of '.' character	5: 0.0494	7 : 0.0410	9 : 0.0401
content-encoding gzip	6: 0.0383	6 : 0.0489	8 : 0.0443
<a href> relative	7 : 0.0379	21 : 0.0179	13 : 0.0269
Count of <meta> tag	8 : 0.0322	4 : 0.0590	7 : 0.0499
<a href> OoD	9 : 0.0301	17 : 0.0212	6 : 0.0499
server apache	10 : 0.0242	20 : 0.0182	16 : 0.0227
Count of 'z' character	11 : 0.0236	25 : 0.0145	21 : 0.0138
<link type="text/css">	12 : 0.0235	9 : 0.0348	10 : 0.0383
<img src="http*">	13 : 0.0215	24 : 0.0149	32 : 0.0073
Count of 'i' character	14 : 0.0196	15 : 0.0265	22 : 0.0137
Count of <p> tag	15 : 0.0190	18 : 0.0193	14 : 0.0255
push()	16 : 0.0178	14 : 0.0273	15 : 0.0236
Count of 'l' character	17 : 0.0168	11 : 0.0318	34 : 0.0070
url extension is .com	18 : 0.0158	22 : 0.0162	23 : 0.0137
Count of 'y' character	19 : 0.0155	34 : 0.0033	30 : 0.0079
vary user-agent	20 : 0.0154	27 : 0.0121	27 : 0.0109
Total href attributes	21 : 0.0146	10 : 0.0339	3 : 0.0739
<a href="https*">	22 : 0.0140	30 : 0.0086	11 : 0.0337
<link href> OoD	23 : 0.0136	8 : 0.0371	12 : 0.0278
cache-control max-age	24 : 0.0130	13 : 0.0276	18 : 0.0187
Count of 'p' character	25 : 0.0124	32 : 0.0053	26 : 0.0114
<form action="http*">	26 : 0.0121	26 : 0.0132	33 : 0.0071
Count of <a> tag	27 : 0.0115	16 : 0.0250	19 : 0.0186
transfer-encoding chunked	28 : 0.0113	12 : 0.0286	17 : 0.0224
Count of 'f' character	29 : 0.0102	29 : 0.0092	28 : 0.0099
<script async="true">	30 : 0.0090	23 : 0.0160	20 : 0.0160
via 1.1	31 : 0.0089	28 : 0.0120	24 : 0.0129
<a href="http*">	32 : 0.0087	33 : 0.0042	31 : 0.0077
Count of tag	33 : 0.0069	19 : 0.0187	25 : 0.0114
<iframe src="*..html">	34 : 0.0063	31 : 0.0083	29 : 0.0081

The total number of HTML tags and the URL lengths were consistent across the sampling scenarios and accounted for an average 22.7% of total feature importance. Half of the features (17 of 34) were from the webpage content and accounted for 43% of total feature importance, and, of the webpage content features, only one feature – counts of the `push()` method – was a JavaScript feature. Eleven of the 34 features were URL features that accounted for 44% of total feature importance, with the remaining six features being HTTP headers that accounted for 13% of total feature importance. Four of the top five

features with the highest average importance were URL features (URL length, total extensions in the URL, the count of "w" characters, and the count of "." characters), accounting for approximately 31% of total feature importance. Nine of the features were HTML features that represented resources via the `src` or `href` attributes or other attributes that can specify resources. They accounted for 17% of total feature importance. Twelve of the 34 features, accounting for 17% of total feature importance, have not been studied for their role in malicious website detection.

The total number of HTML tags was the most important feature and was part of a webpage's *content complexity* [225]. More content requires additional analysis of whether the webpage is malicious and provides added opportunities for placing malicious content inside the webpage. For example, a webpage consisting only of text will not cause a drive-by download, whereas a page with various links, JavaScript, and other resources such as `<iframe>`s, may enable a drive-by download. The next feature identified – URL length – is one that has been frequently used in prior works. This feature was not surprising since attackers use "tiny" as well as longer URLs [226].

URL features, especially the counts of the respective characters in the URL, were observed to be helpful for detecting malicious websites. The count of "w" characters, along with counts of "z," "I," "l," "y," and "p" characters, all appeared on our list. There were no specific known associations between these characters and malicious URLs, though all of the characters, with the exception of "i" and "l," are infrequently used in the English language. The characters "z," "y," "p," and "w" occur 0.27%, 1.77%, 3.16% and 1.28% of the time, respectively [227]. The characters "i" and "l," however, have been prevalent in Kwyjino malware [209]. Furthermore, features derived from character

counts, including ratios, distance vectors, and similarity, have been studied to detect bots and C2 URLs [52] when used in conjunction with unsupervised learning techniques such as clustering or anomaly detection. Lastly, character counts and related metrics were necessary for further study of URLs generated by DGAs. Hence, character counts (and features derived from them) were relevant.

The first HTTP header features we observed have been used in specific attacks and included features that describe how data is packaged in a `response` and how long data should be kept by a client. The `gzip` `content-encoding` header extracted from the HTTP `response` has legitimate uses, but it can be used to evade network scanners [228]. `Chunked` responses, which also appeared on the list, are similar in that they are specified in a `response` header, they have legitimate purposes, and they can be used by attackers [228]. The `cache-control` header, studied by [49], and `max-age` directive can be used by attackers to require a cache to hold a malicious `response` for a long period, thereby enabling a cache-poisoning attack [229].

Other HTTP headers appeared on the list and were the `server` header with a value of `apache`. Apache servers have had many well-known security issues with some enabling backdoors (control) for attackers [230]–[234]. The `via` [206] header is added by proxies that have legitimate uses, but also are well-known to be used by attackers [234]-[235]. The `vary` header can be used by `request`s or `response`s. When used by `response`s with a value of `user-agent`, it specifies whether `response`s will be cached based on the `user-agent` string. This feature is well-known for being misunderstood and misused by developers [236].

We also observed many webpage content features related to links and URLs on the webpage. The total number of relative links or resources (as opposed to absolute links) that point to resources within the page (or relative to the page) is one of the higher ranked features, is an extension of features used in [47], and is known to be used by attackers. It is also a measure of *content-complexity* [225]. Using relative links has the advantage of decreasing the chances of detection for attackers. For example, if the webpage is fetched and successfully loaded, it has gotten past some network-level defenses or other security solutions [237]. Furthermore, relative URLs are known to be leveraged by attackers. Recently, attackers have used relative URLs [237] to bypass Microsoft's ATP for phishing detection. Additionally, the number of OoD links on the page (URLs that are out of the current domain) ranked as a feature. The more links on a webpage, the more opportunities for potential malicious URLs, only one of which must be effective to cause an infection. We also found that the structure of the URLs on the webpage were present in our features. For example, we observed that the protocol (`http` vs `https`) for certain resource references (such as image, links) helped with detection, though they were not highly ranked. The `<iframe>` is well-known for its ability to detect drive-by downloads [59] and we observed that `.html` files in the `<iframe>` `src` attribute made the list.

The next webpage content features deal with specific tags (elements on the webpage). The `<meta>` tag is known to be associated with malicious redirects [238]. Two other tags that appeared were counts of `<p>` and `<center>` tags. Both are formatting tags that specify how text should be rendered. Although they have no known relation to malicious websites, the counts may provide additional insight as to the level of

care that attackers take in formatting their text, information that would be of interest for phishing detection. The other feature, `<link type>`, identifies references to `.css` resources. Although this has legitimate uses, CSS files have also been used for attacks [239].

We observed that one JavaScript feature – counts of the `push()` method – made our list. JavaScript methods can identify obfuscated JavaScript, but the `push()` method is not highly related to malicious JavaScript. Although it made our list, it did not rank highly and it should be noted that although this study performed static analysis, dynamic analysis has been shown to be better suited for malicious code detection. In fact, most prior research required de-obfuscation before analysis of code, as in [65] and [68]). Although we found just one count of a JavaScript method, the `async=true` attribute on the `<script>` tag is a potential attack vector (it instructs the browser to continue rendering third party libraries in the JavaScript).

We then performed feature ranking of the 99 *a priori* features. Webpage content features accounted for 40% of total feature importance, while URL features accounted for 48% of total feature importance, and HTTP features accounted for 12% of total feature importance. Whereas the top two features in Table 7-1 were consistent, none of the top features in Table 7-2 were consistent. Also, we observed that some *a priori* features had little to no importance in our study. Full rankings for the 99 *a priori* features in sampling scenarios are in Table 7-2.

Table 7-2.
The Importance of 99 Features from Prior Research Was Inconsistent Across Sampling Scenarios

99 Features from Prior Research Ranked			
Feature	***No-sampling***	***Over-sampling***	***Under-sampling***
Total HTML Tags	1 : 0.1159	6 : 0.0382	1 : 0.1358
Total Extensions in URL	2 : 0.0957	3 : 0.0520	5 : 0.0493
URL Length	3 : 0.0926	4 : 0.0499	3 : 0.0520
Count of 'w' character	4 : 0.0585	5 : 0.0493	4 : 0.0499
Count of '.' Character	5 : 0.0460	9 : 0.0229	11 : 0.0215
<a href> OoD	6 : 0.0291	15 : 0.0189	6 : 0.0382
Total href attributes	7 : 0.0283	10 : 0.0219	2 : 0.0559
server apache	8 : 0.0261	11 : 0.0215	10 : 0.0219
content-encoding gzip	9 : 0.0255	2 : 0.0559	7 : 0.0372
Count of <link> tag	10 : 0.0241	19 : 0.0175	12 : 0.0196
Count of <meta> tag	11 : 0.0221	7 : 0.0372	8 : 0.0229
Total TLDs in URL	12 : 0.0207	33 : 0.0090	20 : 0.0172
content-language text/html	13 : 0.0195	18 : 0.0183	13 : 0.0194
Count of 'z' character	14 : 0.0190	23 : 0.0151	19 : 0.0175
<link href> OoD	15 : 0.0153	20 : 0.0172	22 : 0.0165
Count of <a> tag	16 : 0.0144	12 : 0.0196	9 : 0.0229
Count of 4-character words	17 : 0.0142	27 : 0.0125	23 : 0.0151
Count of 'y' character	18 : 0.0141	49 : 0.0046	56 : 0.0034
url extension is .com	19 : 0.0135	41 : 0.0055	26 : 0.0133
Total <img src>	20 : 0.0134	16 : 0.0187	15 : 0.0189
content-length	21 : 0.0134	36 : 0.0082	17 : 0.0187
<form action> OoD	22 : 0.0133	22 : 0.0165	21 : 0.0168
Count of tag	23 : 0.0130	25 : 0.0134	14 : 0.0191
cache-control max-age	24 : 0.0128	21 : 0.0168	16 : 0.0187
Count of 'i' character	25 : 0.0128	8 : 0.0229	24 : 0.0147
Count of 'l' character	26 : 0.0122	14 : 0.0191	49 : 0.0046
Count of <style> tag	27 : 0.0113	58 : 0.0031	34 : 0.0090
<script src> OoD	28 : 0.0090	46 : 0.0049	18 : 0.0183
Count of 'p' character	29 : 0.0090	26 : 0.0133	27 : 0.0125
Count of <title> tag	30 : 0.0087	1 : 0.1358	41 : 0.0055
Count of 'f' character	31 : 0.0084	30 : 0.0104	31 : 0.0101
<script type=text/javascript>	32 : 0.0077	37 : 0.0073	28 : 0.0120
Count of 'd' character	33 : 0.0076	17 : 0.0187	35 : 0.0085
Count of 's' character	34 : 0.0074	28 : 0.0120	32 : 0.0092
Count of 5-character words	35 : 0.0072	38 : 0.0066	62 : 0.0021
Count of 'o' character	36 : 0.0068	29 : 0.0114	44 : 0.0050
cache-control no-store	37 : 0.0063	57 : 0.0034	29 : 0.0114
url extension is .i	38 : 0.0063	40 : 0.0063	46 : 0.0049
replace()	39 : 0.0063	34 : 0.0090	40 : 0.0063
Count of 'u' character	40 : 0.0060	43 : 0.0051	30 : 0.0104
Count of <img> tag	41 : 0.0059	61 : 0.0027	36 : 0.0082
<img srcset> OoD	42 : 0.0059	39 : 0.0065	45 : 0.0050
Count of 'b' character	43 : 0.0058	32 : 0.0092	60 : 0.0028
Count of 'e' character	44 : 0.0057	45 : 0.0050	58 : 0.0031
addEventListener()	45 : 0.0054	44 : 0.0050	33 : 0.0090
<base href> OoD	46 : 0.0042	76 : 0.0004	74 : 0.0006
Count of 't' character	47 : 0.0042	59 : 0.0030	48 : 0.0049
Count of <iframe> tag	48 : 0.0041	63 : 0.0021	55 : 0.0037
Count of 'x' character	49 : 0.0040	70 : 0.0010	72 : 0.0007
Count of 'c' character	50 : 0.0039	47 : 0.0049	47 : 0.0049
Count of 'r' character	51 : 0.0037	51 : 0.0045	65 : 0.0014
Count of 'm' character	52 : 0.0037	55 : 0.0037	52 : 0.0045
Count of 'h' character	53 : 0.0036	35 : 0.0085	66 : 0.0013
Count of 'a' character	54 : 0.0032	13 : 0.0194	61 : 0.0027
server nginx	55 : 0.0031	54 : 0.0039	38 : 0.0066
createElement ()	56 : 0.0031	24 : 0.0147	25 : 0.0134
Count of 'n' character	57 : 0.0029	52 : 0.0045	50 : 0.0045
cache-control no-cache	58 : 0.0029	56 : 0.0034	57 : 0.0034
Count of 6-character words	59 : 0.0028	31 : 0.0101	63 : 0.0021
Count of 7-character words	60 : 0.0027	64 : 0.0019	64 : 0.0019

99 Features from Prior Research Ranked			
Feature	***No-sampling***	***Over-sampling***	***Under-sampling***
Count of <input> tag	61 : 0.0023	65 : 0.0014	43 : 0.0051
Count of 'k' character	62 : 0.0022	66 : 0.0013	67 : 0.0012
Count of 'g' character	63 : 0.0021	50 : 0.0045	37 : 0.0073
<img src> OoD	64 : 0.0019	62 : 0.0021	42 : 0.0053
Count of '-' character	65 : 0.0019	42 : 0.0053	39 : 0.0065
Count of 'v' character	66 : 0.0016	69 : 0.0010	59 : 0.0030
write()	67 : 0.0015	74 : 0.0006	69 : 0.0010
cache-control must-revalidate	68 : 0.0014	48 : 0.0049	54 : 0.0039
Count of 'j' character	69 : 0.0013	71 : 0.0009	68 : 0.0011
Count of 8-character words	70 : 0.0012	72 : 0.0007	73 : 0.0007
cache-control private	71 : 0.0012	53 : 0.0041	53 : 0.0041
substring()	72 : 0.0011	80 : 0.0002	70 : 0.0010
url extension is .net	73 : 0.0011	67 : 0.0012	71 : 0.0009
<iframe src> OoD	74 : 0.0010	60 : 0.0028	51 : 0.0045
escape()	75 : 0.0005	79 : 0.0002	75 : 0.0005
cache-control public	76 : 0.0004	68 : 0.0011	78 : 0.0003
setTimeout()	77 : 0.0004	84 : 0.0001	81 : 0.0002
parseInt()	78 : 0.0003	78 : 0.0003	77 : 0.0004
concat()	79 : 0.0003	73 : 0.0007	85 : 0.0001
Count of <frame> tag	80 : 0.0002	83 : 0.0002	80 : 0.0002
<frame src> OoD	81 : 0.0002	81 : 0.0002	79 : 0.0002
unescape()	82 : 0.0002	82 : 0.0002	76 : 0.0004
exec()	83 : 0.0002	75 : 0.0005	84 : 0.0001
fromCharCode()	84 : 0.0001	88 : 0.0001	83 : 0.0002
Count of <object> tag	85 : 0.0001	89 : 0.0001	87 : 0.0001
<area href> OoD	86 : 0.0001	77 : 0.0004	89 : 0.0001
<embed src> OoD	87 : 0.0001	85 : 0.0001	82 : 0.0002
eval()	88 : 0.0001	87 : 0.0001	91 : 0
search()	89 : 0.0001	91 : 0	93 : 0
Count of <embed> tag	90 : 0	90 : 0	90 : 0
charCodeAt ()	91 : 0	95 : 0	86 : 0.0001
<object data> OoD	92 : 0	92 : 0	88 : 0.0001
hidden <iframe>	93 : 0	86 : 0.0001	92 : 0
setInterval()	94 : 0	94 : 0	96 : 0
<source src> OoD	95 : 0	93 : 0	94 : 0
<source srcset> OoD	96 : 0	97 : 0	95 : 0
link()	97 : 0	98 : 0	97 : 0
<audio src> OoD	98 : 0	96 : 0	98 : 0
<video src> OoD	99 : 0	99 : 0	99 : 0

7.5.3 **Sampling Scenarios**

In the no-sampling scenario, we observed that two out of the nine models improve when using *discovered* features versus *a priori* features. The average accuracy, AUC, MCC, Precision, and Recall changed by -0.03%, 0.0045, -0.001, -0.013 and 0.01, when using *discovered* versus *a priori* features respectively. In the no-sampling scenario the *discovered* feature set performs nearly as well as the *a priori* feature set albeit with 66% fewer features. Results are shown in Figure 7-4, and full results shown in Appendix D.

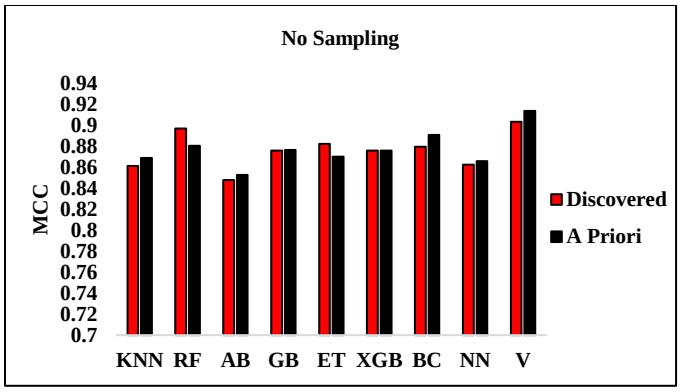


Fig. 7-4. Discovered features performed approximately as well as the prior features in the no-sampling scenario

In the over- and under-sampling scenarios, we observed that three and two out of the nine models improved with *discovered* features versus *a priori* features, respectively. The average accuracy, AUC, MCC, Precision, and Recall changed by -0.2%, 0.006, -0.007, -0.027, 0.016, in the over-sampling scenario and -0.37%, -0.0008, -0.01, -0.019, 0.0049 in the under-sampling scenario with *discovered* features. Hence, we observed similar behavior as in the no-sampling scenario. Results are shown in Figure 7-5 and Figure 7-6, with full results shown in Appendix D.

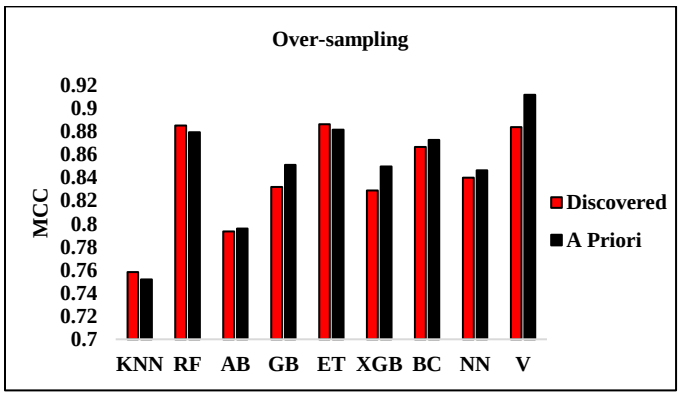


Fig. 7-5. Discovered features performed approximately as well as the a priori features in the over-sampling scenario

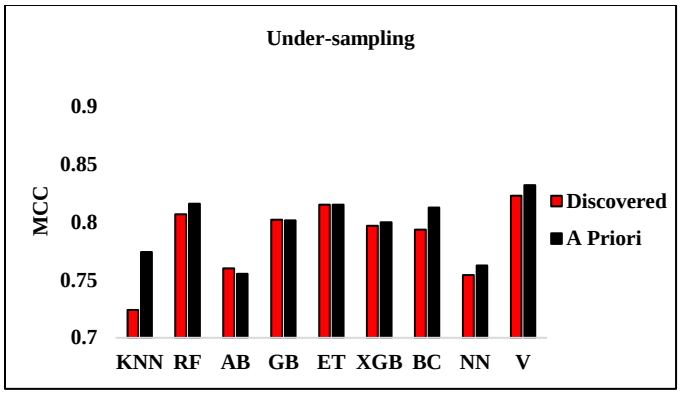


Fig. 7-6. Discovered features performed approximately as well as the a priori features in the under-sampling scenario

7.5.4 **Feature Transformation**

We now discuss the results of using features created through transformation of the features and the components created by performing FT w/FS and FT w/PCA on these features. For the transformed features (2,278 features were created from the 34 *discovered* features, and 19,503 from the 99 *a priori* features) we observed a change in accuracy in AUC, MCC, Precision, and Recall of 0.23%, 0.0113, 0.0129, -0.0006, 0.0231, respectively, and all nine models improved when using features created from *discovered* features versus *a priori* features – 88% fewer created features.. The effect of the feature transformation with feature selection is shown in Figure 7-7 and full results are found in Appendix D.

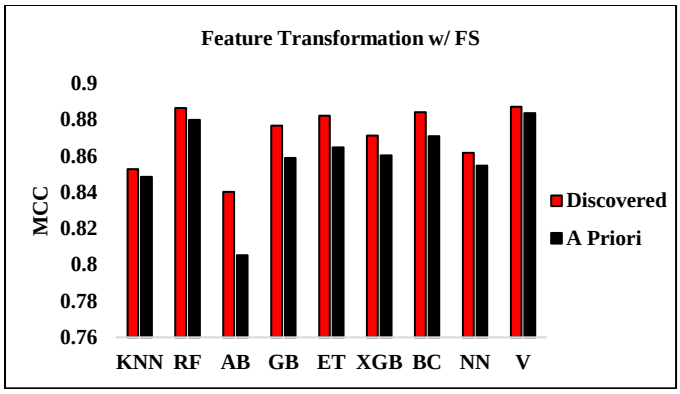


Fig. 7-7. Discovered features out-performed the a priori features in the feature transformation with feature selection

The discovered features also appeared to produce better components from PCA and did so with fewer features. One hundred and twenty-five components were created from the 34 discovered features with PCA, compared to 750 components created from the 99 *a priori* features. With components from discovered features compared to the components created from *a priori* features, we observed a change in ACC, AUC, MCC, Prec, and Rec of 7.65%, 0.0743, 0.1099, 0.0789, 0.0658, respectively, and eight of the nine models demonstrated overall improvement compared to using models built with the components from the *a priori* features. The effect of the feature transformation with PCA is shown in Figure 7-8, with full results shown in Appendix D.

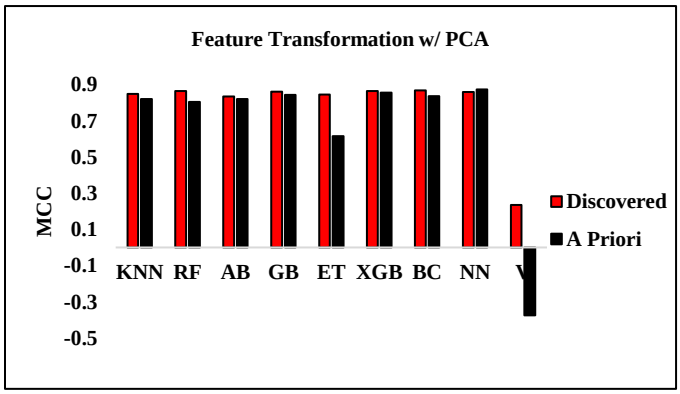


Fig. 7-8. Discovered features out-performed a priori features in the feature transformation with PCA

7.5.5 **Hyperparameter Tuning and Cross-Validation**

We tuned the best performing model in each scenario built from an 80:20 split of train to test data (no sampling, over-sampling, under-sampling, FT w/FS, and FT w/PCA). In this section, we compare the results of the tuned models to their respective non-tuned counterparts. With tuning, only one of the models built with the 34 *discovered* features achieved a higher MCC, and the average MCC of the tuned models was 0.004 less than the average MCC of the non-tuned models. When performing hyperparameter tuning and cross-validation on the models built from 99 *a priori* features, we observed that the average MCC improved by 0.007 compared to the non-tuned models and all five of the best performing models improved.

For further assurance that our results were not a product of our initial 80:20 split, we repeated our approach with a 70:30 split of training to testing data. We observed that the average MCC increased by 0.011 when tuning, and four of the five best performing models improved when tuning models built with the 34 *discovered* features compared to their non-tuned counterparts. When performing hyperparameter tuning and cross-validation on the models built from 99 *a priori* features, we observed that the average MCC improved by 0.011, and all five of the best performing models improved. Full results are available in Table 7-3 which show the model used, scenario, MCC, and scoring metric used during tuning achieve the respective MCC. Results from the *a priori* features are separated by a "/" from the results of the *discovered* features.

Table 7-3.
Hyperparameter Tuning and Cross-Validation
Slightly Improved Detection Performance for
Discovered and A Priori Features

Cross-Validation and Tuning for A Priori / Discovered			
Model	***Scenario - Split***	***MCC***	***Scoring Metric***
BC **RF**	No - 70:30	0.9027/ 0.8926	precision weighted balanced accuracy
RF **RF**	Over - 70:30	0.8802/ 0.8830	precision weighted precision weighted
BC **ET**	Under - 70:30	0.8113/ 0.7967	precision weighted recall weighted
BC **BC**	FT w/ FS - 70:30	0.8865/ 0.8920	precision weighted precision weighted
NN **NN**	FT w/ PCA - 70:30	0.8853/ 0.8744	recall recall
BC **RF**	No - 80:20	**0.9066/** 0.8942	precision weighted precision weighted
RF **ET**	Over - 80:20	0.8819/ 0.8718	precision weighted precision weighted
RF **ET**	Under - 80:20	0.8174/ 0.8049	precision weighted recall weighted
RF **RF**	FT w/ FS - 80:20	0.8841/ 0.8843	precision weighted precision
NN **BC**	FT w/ PCA - 80:20	0.8863/ 0.8792	recall precision weighted

We then performed hyperparameter tuning and cross-validation of the voting classifier. Full results for each scenario are shown in Table 7-4.

Table 7-4.
Hyperparameter Tuning and Cross-Validation Slightly
Improved Detection Performance for Discovered and
A Priori Features

MCC of Tuned Voting Classifier in Different Scenarios					
Scenario	***No***	***Over***	***Under***	***FT w/ FS***	***FT w/ PCA***
70/30 Discovered	0.9144	0.9066	0.8231	0.8966	**0.2406**
70/30 Prior	**0.9281**	0.9192	**0.8375**	0.8911	-0.4314
80/20 Discovered	0.9174	0.9071	0.8177	**0.8973**	-0.0275
80/20 Prior	0.9264	**0.9226**	0.8295	0.8884	-0.3831

Hyperparameter tuning and cross-validation showed that the default parameters for all models, except the voting classifier, performed similarly to the tuned classifiers. The voting classifier improved when we performed tuning (except when used with PCA).

We observed that while the discovered features performed better with default parameters, they performed slightly worse during hyperparameter tuning.

7.5.6 RQ5: Is Feature Discovery Feasible for Malicious Website Detection?

In RQ5 we examined the detection ability of *discovered* features. With the *discovered* features we can obtain a best classifier performance of an MCC 0.7043 using unsupervised learning techniques, and 0.9174 using supervised learning techniques. The unsupervised results were not promising however unsupervised techniques are not as common for detecting malicious websites (except for C2 traffic). The supervised results suggested that a discovery approach can be used however, for further insight into their feasibility, we need to compare our approach and results with prior research that uses the notion of *a priori* features.

Ma in [35] leveraged URL and host-based (DNS queries, WHOIS properties, and geographic information) and incorporated online learning in [36]-[37] and an SVM classifier to identify malicious URLs. Ma used strictly URL and host-based features. We both investigated different training scenarios however Ma focused on training an SVM several times with online learning while we focused on sampling, feature transformation, and evaluated nine different algorithms. Additionally, we both accepted our malicious dataset from external parties though their dataset consisted of millions of URLs while ours included ~47000 entries. Given the design of our experiments we also quantified best-case performance differently – Ma with error rate (best-case 2.6%) while we focused on MCC (best-case 0.9174) due to our dataset imbalance. Whittaker [42] also took an online approach and trained and evaluated an LR classifier on millions of webpages (with different algorithms), focused on phishing, and presented their results as a tradeoff

between precision and recall with their best performance being a precision and recall over 0.95 respectively. Prophiler [45] also focused on classifying a large number of malicious webpages with an evaluation dataset of almost 19 million webpages and achieved a false positive and false negative rate of 9.88% and 0.77% with HTML, JavaScript, URL, and host features. They used naïve Bayes, random forest, logistic regression, and other decision tree algorithms.

CANTINA+[46], like us, extracted webpage features and investigated several learning algorithms -support vector machines, logistic regression, Bayesian networks, J48 decision tree, random forest, and adaboost. Unlike us and like Ma, they relied on external sources (for example Page Rank) for features which differed from our approach of only using web `response` features. Their study included two phishing sets of 1,595 and 624 webpages respectively. They achieved a true positive and false positive rate of 4.24% and 1.948% respectively.

Marchal [91] detected phishing with a GB (provided by `Scikit` Learn) classifier with high performance metrics (an AUC up to 0.999 compared to our AUC of 0.9464) and differentiated themselves by detecting phishing webpages in different languages. They selected 212 features (including the URL and webpage content) for detection, many of which overlap with other prior works. Their dataset consisted of 150,000 legitimate phishing URLS and 3,366 phishing URLs. Like our study, they performed cross-validation.

Phishmon [83] achieved an accuracy of 95.4% (compared to our accuracy of 98.38%) with a false positive rate of 1.3% on a dataset of 17,508 legitimate and 4,807 phishing webpages. Their approach, unlike other approaches and like our approach,

incorporated all HTTP headers in conjunction with webpage content features for detection. They used four different classifiers (classification and regression trees, k nearest neighbor, adaboost, and random forest) and provided a notion of feature importance.

BINSPECT [47] who has a similar approach (aside from feature discovery) to our own used several machine learning classifiers (J48, random tree, random forest, naïve bayes, bayesian networks, support vector machine, and logistic regression) and some of our *discovered* features overlap with their features. However, our study discovered all our features and did not require external sources such as search engine results. We also differed in that our dataset contained C2 URLs while theirs did not. Our accuracies were similar – their accuracy is 97.81% compared to our accuracy of 98.38%. In addition, we observed that 22 of the 34 features *discovered* have been used in prior research. Additionally, most of the features in the *discovered* list have some known association with attacks or malicious techniques. Cova [65] leveraged *a priori* features that can identify malicious or suspicious JavaScript and anomaly detection to create JSAND. They evaluated their approach on 823 samples from four different data sources and achieved a false negative rate of 0.2%. Xu [49] took the closet approach to our own regarding the features for detection. However, they also included other features like network traffic summaries, which required additional overhead, and they performed feature selection from their *a priori* features for detection. Their approach evaluated four different classification algorithms - naïve Bayes, logistic regression, support vector machines, and J48 and achieved a best-case 99.986% accuracy and they noted the five most selected features.

Basnet [50] used correlation-based feature selection and wrapper feature selection to find relevant features among 177 URL, webpage content, and search engine features along with a naïve Bayes, logistic regression, and random forest classifier. They observed wrapper-based feature selection techniques could improve false negative rates by 44.5% while we found that feature selection generally decreased classifier performance. Li [218], like our study, performed feature transformation during their detection of malicious URLs and did so on seven domain-based, 21 host-based, six reputation-based, and 28 lexical features. Their goal was to demonstrate the benefit of feature transformation when used with different algorithms and noted that feature engineering improved detection rates from 68% to 86%, 58% to 81% and 63% to 82% for KNN, SVM, and NNs classifiers respectively. Their best accuracy was measured at 97.80%.

There are similarities are well as differences between our approach and results from those of prior research. First, we note that our results are comparable (and often better) than those from prior research however true comparison is difficult. Marchal [91] and Xu [49] have achieved highly accurate results with *a priori* features however this was done on phishing alone in the case of Marchal, and with additional features like network traffic statistics as in the case of Xu. Marchal and Xu provided better results than any prior research we have encountered. Second, we observed that features derived from a web `response` are simpler to gather. Page rank, WHOIS information, network traffic statistics all require additional instrumentation and overhead. Based on our detection metrics as well as on comparisons to prior research, we postulate that feature discovery is feasible for malicious website detection.

7.5.7 RQ6: How do Discovered Features' Detection Ability Compare to Those from Prior Research?

With RQ6, we compared the ability of the *a priori* features from prior research to the ability of those found via our discovery approach. In the sampling scenarios, we saw little change in detection performance when using discovered features versus *a priori* features. However, we did obtain similar detection metrics with fewer features. With feature transformation, the *discovered* features outperformed the *a priori* features. Hence, we postulated that discovered features can be used to create better transformed features for detection and also require fewer features for detection. During tuning of the models, we noted that the *a priori* features slightly outperformed the discovered features. Thus, we answered RQ6 in a mixed fashion. The *discovered* features performed nearly as well as the *a priori* features, with slight differences depending on the scenario, but they did so with fewer features.

7.5.8 RQ7: Can a Discovery Approach be Applied to Several Threats when Only Features from a Web Response are Available?

With RQ7, we explored whether the discovery approach can be applied to a set of several threats with a limited number of features (those that can be gathered from the `response` to web `request`). We designed our experiment to simulate the real-world constraints by using a dataset consisting of several threats and by leveraging techniques from prior research. Our limited insight into these threats (we did not hand select them nor were they homogenous) also simulated real-world constraints. In addition, we included C2 URLs in our dataset, an element that is often not absent from other studies that detected multiple types of threats. To examine this research question, we looked

further into the performance metrics. Overall, our ACC, AUC, and MCC performed well and were comparable to (and sometimes exceeded) the accuracy of other approaches. However, our findings do suggest that this approach alone is not enough. To further examine whether this approach can be a supplement to other detection capabilities, we examined the FPR of our models since a large number of false positives [240] poses a challenge for practical detection solutions. The FPR of our best performing model in the no-sampling scenario was 0.3% (the tuned voting classifier) and our worst performing classifier in the no-sampling case (AB) had an FPR of 1.326% which bodes well for inclusion into a practical solution. Furthermore, the features extracted in this approach can be extracted from a web `request` `response` and can be added to other security solutions. As a result, we postulated that a discovery approach, while not sufficient in isolation, can be used as a supplement to other detection techniques.

7.6 Conclusions

We performed a comprehensive evaluation of discovering features for malicious website detection. We built two unsupervised learning models and nine supervised detection models over various sampling and feature transformation scenarios. Based on our study, we postulated that discovering features (versus selecting features *a priori*) was feasible and performed nearly as well as the features from prior research, but did so with fewer features.

Chapter 8: Evaluation on an Additional Dataset

8.1 Introduction

Feature-based malicious website detection has shown promise in prior research as well in our studies (see Chapters 4–7). Thus far, our experiments gathered a dataset (Dataset 1) and split it into two portions – a training portion and a testing portion. We were able to achieve high performance metrics in Chapter 7, with MCCs of up to 0.9281 but we performed our study on a single dataset that was created from a single point in time. Although this approach is common, it leads to a lack of insight into how the models built (and their features) would perform on additional datasets that may have been gathered at another point in time. In an operational scenario, a detection model must work on different datasets regardless of time of collection or dataset source. Using a single dataset for training and testing provided limited intuition regarding the feasibility of training models using features and applying it to an operational scenario.

In this chapter, we explore the application of the models and features identified in Chapters 4–7 to a different dataset. This new dataset (Dataset 2) differs from the dataset used in Chapters 4–7 in three ways: 1) the benign portion consists of more entries, 2) the malicious dataset is derived from another source, and 3) the dataset was collected at a different point in time. The first dataset, referred to as Dataset 1 and used in Chapters 4–7, consisted of domains gathered from the top 39,877 websites in the Alexa Top 1M and 6,894 websites provided by Cisco Talos. We collected Dataset 1 in August of 2018. The second dataset, referred to as Dataset 2, consisted of websites from the Alexa Top 1M collected four months later in December of 2018. We segmented this dataset into two portions – benign websites and malicious websites. In the malicious portion of the

dataset, we grouped those websites in the Alexa Top 1M that appeared in threat intelligence information from Cymon.io [193]. We created the benign dataset from websites in the Alexa Topo 1M that did not appear in the Cymon.io threat intelligence information. Throughout this chapter, we report the various experiments performed on Dataset 2 with the goal of better understanding how the models and features discovered in Chapters 4–7 would perform on an additional dataset. In this portion of the study, we made the following contributions:

- We demonstrated that the 34 features identified in Chapter 7 served as a foundation for detection, but required adjustments in order to be effective;
- We compared features for detection over two datasets gathered from different sources at different points in time; and
- We identified two additional features that greatly improved detection on another dataset.

8.2 Related Research

Ma et al. [35] used two different benign sources (Yahoo! and DMOZ) and two different malicious sources (PhishTank and Spamscatter). From the benign and malicious datasets, the authors created four datasets in which the benign and malicious dataset combinations were Yahoo-PhishTank, Yahoo-Spamscatter, DMOZ-PhishTank, and DMOZ-Spamscatter. They trained an LR classifier on each set and evaluated the model on each set. They received low error rates (0.9%, 1.24%, 1%, and 3.01%) when training and testing on the same dataset, but observed error rates of up to 44% when training and evaluating on a different dataset. They repeated this approach of training and evaluating on different datasets in subsequent research [64], [159], [161]. JSAND creators [65]

accomplished a similar goal in their evaluation. They trained their models on two known datasets: 1) “known-good,” consisting of webpages from Google, Yahoo, and Alexa with malicious websites removed; and 2) “known-bad,” consisting of URLs from datasets used in prior research (spam trap, SQL injection, malware forum, and wepawet). Although they did not focus on evaluating a separate dataset, they identified 137 URLs as malicious (on the separate dataset), with 15 being false positives. Le et al. [63] trained their detection mechanism on a group of malicious websites and evaluated them on another set of benign websites and malicious websites. Blum et al. [38] trained their models on University of Alabama phishing websites and evaluated them on other feeds from Cyveillance, observing error rates as high as 30% when the training sets and testing set were from different sources. Researchers [160] also provided a training and testing dataset from different sources.

He et al. [43] built their dataset with the combination of websites from multiple sources – Alexa, 3Sharp, and Phishtank – but they evaluated their detector on two datasets. They derived the datasets from the same source, but collected them at different points in time. They observed that their detector performed well, with a TPR of 97% and FPR of 4%. CANTINA+ authors [46] conducted several experiments on phishing webpages, including collecting two datasets of phishing websites from the same source, achieving a TPR of 93.47% and FPR of 0.608%.

Prior researchers have reported mixed results. Some have observed similar performance when applying their methods to other datasets (either gathered from a different source or collected at a different point in time), while others have seen performance decreases. We made two observations. First, we observed that performance

decreased when researchers trained and tested on datasets composed of different threats ([35], for example), while researchers tended to report consistent performance when focused on detecting one type of threat. Secondly, we observed that researchers were more likely to show consistent performance when data collections occurred closer in time (authors [45] with Prophiler, for example). Based on these observations, we analyzed our detection performance on an additional dataset.

8.3 Research Questions

In this section we list the research questions addressed in Chapter 8

8.3.1 Research Question 8

With this research question, we explored the ability to apply models built from data derived from one dataset to models built from data derived from another dataset. To that end, we examined the performance of the best-performing RF model (built from 34 features as detailed in Chapter 7 and trained on Dataset 1) when evaluated on Dataset 2. We stated RQ8 as follows:

RQ8: How robust are malicious website detection models when applied to a new dataset?

8.3.2 Research Question 9

We crafted this research question to guide our investigation into the effectiveness of the features identified in Chapter 7 and their ability to detect malicious websites on a new dataset. We used a series of feature selection techniques to identify the features noted in Chapter 7, some of which had been used in prior research, while others had not. We re-trained models on our new dataset, but limited our features to the 34 identified in Chapter 7. By doing so, we gained insight into the robustness of the features identified

from our previous work and determined whether they could be applied to additional datasets. RQ9 is stated as follows:

RQ9: How do the features identified perform on a new dataset?

8.3.3 Research Question 10

Although we evaluated the features identified in Chapter 7 and the robustness of the model built in Chapter 7, here we investigated how we could leverage on a new dataset other aspects of the experiment results reported in Chapters 4–7. We aimed to identify which aspects, if any, from our prior experiments could be leveraged on this new dataset. RQ10 is stated as follows:

RQ10: What aspects from prior experiments can we apply to a new dataset?

8.4 Feature Consideration, Dataset, Analysis Approach

8.4.1 Feature Consideration

In Chapter 7, we captured the performance of our detection models constructed with 34 features that we identified through feature selection and with 99 features gathered from prior research. In this chapter, we focus on the 34 features identified in Chapter 7 (referred to as the "identified features"), but expand our analysis to the 99 features gathered from prior research and reported in Chapter 7 (referred to as the "prior features"), as well as to 288 additional features (referred to as the "features after the first feature-selection step"). As reported in Chapter 7, we obtained these 288 features by dropping from our dataset those features that were consistent at least 95% of the time and by dropping from our dataset those features with high VIF values before application of the XGB algorithm.

8.4.2 Datasets

In the study portion reported in this chapter, we continued to make use of Dataset 1, though we focused our evaluations on a Dataset 2 consisting of the Alexa Top 1M websites. We selected as malicious websites those from the Alexa Top 1M that were found in threat intelligence data provided by Cymon.io [193]. For benign websites, we chose those that appeared in the Alexa Top 1M but did not appear in the Cymon.io threat intelligence information. For clarity, we refer to the dataset used in Chapters 4–7 as Dataset 1 and to the new dataset of Alexa/Cymon.io websites as Dataset 2.

8.4.3 Analysis Approach

To explore additional (and larger) datasets, it was necessary for us to perform analysis more efficiently than we had performed the analysis in previous portions of our inquiry. Thus, we narrowed our focus to an RF classifier, which had proven to be the among best performing classifiers in our prior studies and performed well in related research as well. Additionally, we leveraged the `class weight` parameter available in the `SciKit` library, which can be an alternative to over-sampling and under-sampling.

8.5 Results

8.5.1 RQ8: How Robust are Malicious Website Detection Models when Applied to a New Dataset?

We began our investigation by applying to Dataset 2 the RF model built in the no-sampling scenario with the 34 features in Table 7-1.

8.5.1.1 Evaluation on Previous Models

First, we evaluated the performance on the new dataset of our best-performing model from Chapter 7, the RF model. We built this RF model with the 34 features

identified in Chapter 7. Table 8-1 below shows the performance of the RF model on Dataset 2, the new dataset of Alexa Top 1M with Cymon.io [193] threat intelligence data as ground truth.

Table 8-1.
Applying the Best Random Forest Classifier Built in Chapter 7
from Dataset 1 to Dataset 2 Yielded Poor Detection Results

Detecting Malicious Websites in Dataset 2 with a Model Built with Dataset 1						
FPR	***FNR***	***ACC***	***AUC***	***MCC***	***Prec***	***Rec***
0.5599	0.4846	0.4432	0.4778	-0.0179	0.0384	0.5154

As shown in Table 8-1, results demonstrated poor metrics and an inability to classify the new dataset. This observation was similar to that made by Ma et al. [35], who observed high errors when training and testing datasets from different sources. This observation prompted further investigation into the datasets and potential causes. However, from the results in Table 8-1, we observed that we could not directly apply the model derived from Dataset 1 to Dataset 2.

8.5.1.2 Feature Correlation Investigation

To investigate potential causes for the poor performance of our model, we began to examine Dataset 2 and compare it to Dataset 1. We first analyzed the correlation of each variable to the target variable (whether the website is malicious) in order to determine whether there were differences between the respective correlations for the features in Dataset 1 and Dataset 2. We did this on three different sets: the 34 features identified by our research and reported in Chapter 7; the 99 features gathered from prior research and reported in Chapter 7; and the 288 features remaining after removing features that were consistent at least 95% of the time and that had high VIF values before application of the XGB algorithm. By expanding our analysis to the 99-feature and 288-

feature sets, we gained insight into whether there were additional features that had high correlation with the target variable, in case they were applicable to Dataset 1 but not to Dataset 2 or vice versa. Tables 8-2 and 8-3 show the correlation values for the 34 features and the 99 features on Dataset 1 and Dataset 2. Table D-1 in Appendix D shows the correlation values for the 288 features on Dataset 1 and Dataset 2.

Table 8-2.
The 34 Features Identified for Detection in
Chapter 7 Had Different Correlation Values
for Dataset 1 and Dataset 2

Correlation with Maliciousness for Identified on Datasets 1 and 2		
Feature	***Dataset 1***	***Dataset 2***
URL Length	0.5245	0.0188
Count of '.' character	0.5078	0.0159
Total Extensions in URL	0.4672	0.0006
content-encoding gzip	0.4350	0.0027
Count of 'w' character	0.3821	0.0022
Count of 'z' character	0.3129	0.0074
Count of 'y' character	0.2854	0.0103
transfer-encoding chunked	0.2797	0.0102
Count of 'i' character	0.2566	0.0096
Total HTML Tags	0.2370	0.0007
<script async="true">	0.2170	0.0100
Total href attributes	0.1946	0.0001
cache-control max-age	0.1934	0.0108
Count of 'l' character	0.1934	0.0101
Count of <a> tag	0.1867	0.0009
<link href> relative	0.1857	0.0009
<link href> OoD	0.1822	0.0173
<a href> OoD	0.1748	0.0019
<link type="text/css">	0.1704	0.0148
Count of 'f' character	0.1655	0.0057
Count of 'p' character	0.1493	0.0055
<a href="https*">	0.1490	0.0041
<iframe src="*..html">	0.1221	0.0046
url extension is .com	0.1415	0.0205
via 1.1	0.1085	0.0135
Count of <p> tag	0.1048	0.0032
Count of <meta> tag	0.1008	0.0103
<a href="http*">	0.0995	0.0062
<form action="http*">	0.0832	0.0045
server apache	0.0745	0.0043
<img src="http*">	0.0708	0.0095
push()	0.0471	0.0008
Count of tag	0.0335	0.0035
vary user-agent	0.0226	0.0083

Table 8-3.
The 99 Features from Prior Research Had Different Correlation Values
for Dataset 1 and Dataset 2

Correlation Values for 99 Features on Datasets 1 and 2		
Feature	*Dataset 1*	*Dataset 2*
URL Length	0.5245	0.0188
Count of '.' character	0.5078	0.0159
Total Extensions in URL	0.4672	0.0006
content-language text/html	0.4624	0.0080
content-encoding gzip	0.4350	0.0026
Count of 'w' Character	0.3821	0.0022
Count of 'z' Character	0.3129	0.0074
Count of 'y' character	0.2854	0.0103
Count of 'i' character	0.2566	0.0096
Total TLDs in URL	0.2551	0.0303
<script type=text/javascript>	0.2406	0.0174
Count of 4-character words	0.2387	0.0314
Total HTML Tags	0.2370	0.0006
<script src> OoD	0.2170	0.0153
Count of 7-character words	0.2084	0.0019
Count of tag	0.2080	0.0054
Count of <link> tag	0.2059	0.0235
Count of 'e' character	0.2042	0.0073
Count of 'x' character	0.195	0.0062
Total href attributes	0.1946	0.0001
cache-control max-age	0.1934	0.0108
Count of 'l' character	0.1934	0.0101
Count of <iframe> tag	0.1924	0.0020
Count of <a> tag	0.1867	0.0010
Count of 'u' character	0.1860	0.0199
<link href> OoD	0.1822	0.0173
Count of 6-character words	0.1765	0.0057
Count of 's' character	0.1748	0.0001
<a href> OoD	0.1748	0.0019
Count of 'b' character	0.1737	0.0101
Count of 8-character words	0.1725	0.0021
Count of 'n' character	0.1721	0.0081
Count of <img> tag	0.1715	0.0032
Count of '-' character	0.1674	0.3660
Count of 'f' character	0.1655	0.0057
server nginx	0.1624	0.0219
<Total img src>	0.1604	0.0050
Count of 5-character words	0.1519	0.0160
Count of 'p' character	0.1493	0.0055
cache-control must-revalidate	0.1492	0.0084
Count of 't' character	0.1420	0.0037
url extension is .com	0.1415	0.0205
cache-control no-cache	0.1411	0.0103
createElement()	0.1385	0.0039
Count of 'a' character	0.1376	0.0262
Count of <style> tag	0.1351	0.0135
<img src> OoD	0.1347	0.0070
cache-control no-store	0.1291	0.0103
Count of 'h' character	0.1223	0.0080
<iframe src> OoD	0.1216	0.0017
cache-control private	0.1207	0.0089
Count of 'k' character	0.1188	0.0070
Count of 'd' character	0.1176	0.0015
Count of 'r' character	0.1152	0.0146
<form action> OoD	0.1060	0.0016
cache-control public	0.1045	0.0024
Count of <meta> tag	0.1008	0.0104
Count of <input> tag	0.0969	0.0018
addEventListener()	0.0938	0.0093
Count of 'c' character	0.0913	0.0182
Count of 'g' character	0.0821	0.0100
<base href> OoD	0.0804	0.0122
replace()	0.0770	0.0058
content-length	0.0748	0.0040
server apache	0.0745	0.0043
concat()	0.0689	0.0087
Count of 'v' character	0.0651	0.0066
url extension is .net	0.0607	0.0011
Count of 'j' character	0.0572	0.0081
Count of 'o' character	0.0570	0.0192
setTimeout()	0.0516	0.0034
Count of <title> tag	0.0499	0.0066
substring()	0.0497	0.0025
<iframe> is hidden	0.0439	0.0027
exec()	0.0423	0.0023
parseInt()	0.0417	0.0007
escape()	0.0397	0.0196
<img srcset> OoD	0.0394	0.0073
fromCharCode()	0.0368	0.0073
Count of 'm' character	0.0300	0.0110
setInterval()	0.0270	0.0009
eval()	0.0233	0.0046
charCodeAt()	0.0225	0.0059
<source srcset> OoD	0.0189	0.0023
search()	0.0182	0.0012
write()	0.0173	0.0012
unescape()	0.0162	0.0041
<source src> OoD	0.0143	0.0009
Count of <object> tag	0.0138	0.0003
Count of <embed> tag	0.0136	0.0006
<object data> OoD	0.0133	0.0003
<area href> OoD	0.0131	0.0009
<embed src> OoD	0.0093	0.0005
Count of <frame> tag	0.0072	0.0009
link()	0.0069	0.0001
url extension is .i	0.0063	0.0069
<video src> OoD	0.0062	0.0001
<audio src> OoD	0.0042	0.0004
<frame src> OoD	0.0026	0.0012

Table D-1 in Appendix D shows that the count of "-" characters had high correlation with the target variable. From Tables 8-2 and 8-3, we observe that the features that have high correlation with the target variable in Dataset 1 no longer have a high correlation with the target variable in Dataset 2. This suggests there are differences between these datasets and this is one potential cause for the poor performance. We did notice however, that counts of the "-" had high correlation (0.3660) in Table D-1 (Appendix D). This observation is noted for the remainder of this experiment.

8.5.1.3 T-SNE Analysis

Given the poor results in Table 8-1, we also analyzed the distribution of features. We analyzed the distribution of the 34 features used to build the model, the 99 features gathered from prior research, and the set of 288 features remaining after removal of consistent features and high-VIF features. We applied t-distributed stochastics neighbor embedding (t-SNE) [241] (a non-linear, dimensionality-reduction technique that helps visualize high-dimensional data) on the 34 features and on each individual feature category. We used t-SNE, an exploratory analysis technique, to visually compare the features from both sets. We took a sample of 5,000 websites from both datasets in each case. We first performed t-SNE on the 34 features. Results are shown in Figure 8-1.

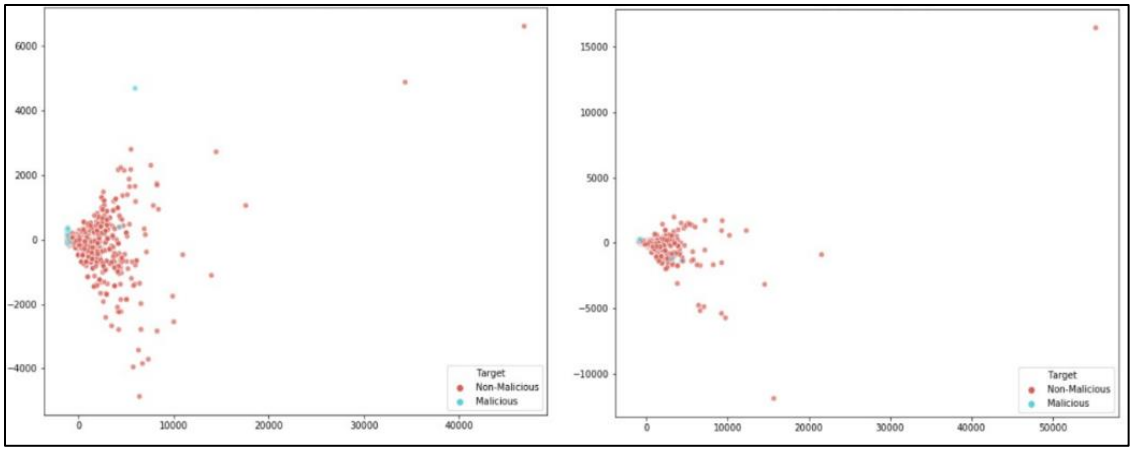


Fig. 8-1. T-SNE analysis performed on the features identified in Chapter 7 from a sample of 5,000 websites from Dataset 1 and Dataset 2 showed no clusters for malicious websites

We then separately performed t-SNE on the webpage content, URL, and HTTP header features. Results are shown in Figure 8-2, Figure 8-3, and Figure 8-4 below.

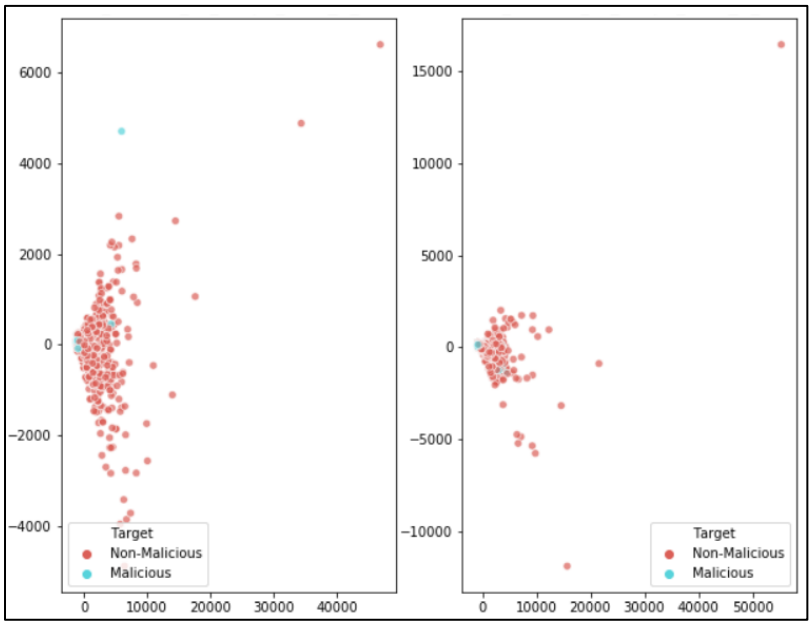


Fig. 8-2. T-SNE analysis performed on the webpage content features collected in Chapter 4 from a sample of 5,000 websites from Dataset 1 and Dataset 2 showed no clusters for malicious websites

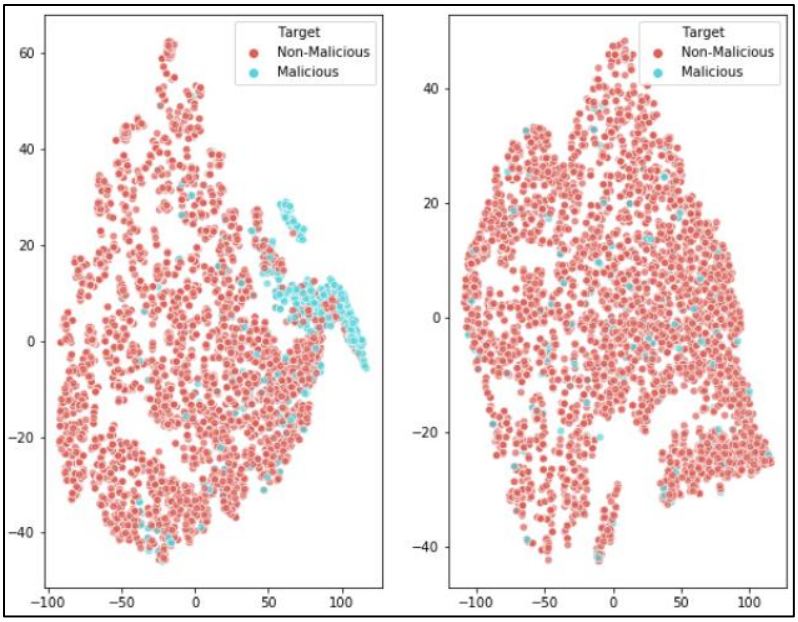


Fig. 8-3. T-SNE analysis performed on the URL features collected in Chapter 5 from a sample of 5,000 websites from Dataset 1 and Dataset 2 showed clusters for malicious websites on Dataset 1

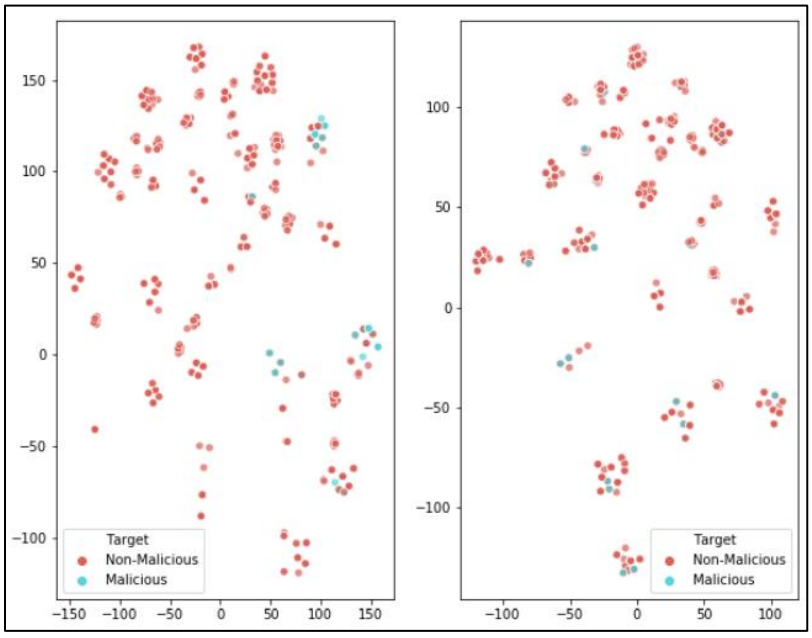


Fig. 8-4. T-SNE analysis performed on the HTTP header features collected in Chapter 6 from a sample of 5,000 websites from Dataset 1 and Dataset 2 showed no clusters for malicious websites

We did not observe any clusters for the malicious websites in Figures 8-1 and 8-2, though we did see a cluster in Figure 8-3 (URL features) for Dataset 1 that was not

present from Dataset 2. Additionally, we observed that URL features (see Chapter 5) produced higher accuracies compared to the other features. We also observed a small cluster in Dataset 1 (see Figure 8-4). These are additional potential explanations for the failure of the model from Chapter 7 to detect malicious websites in Dataset 2.

8.5.1.4 Statistical Tests on Dataset 1 and Dataset 2

We then performed further statistical tests to probe the differences between Dataset 1 and Dataset 2. First, we performed a two-sample KS test [197] to determine whether the 34 features from Datasets 1 and 2 were from the same distribution. The test is more suited for continuous variables and is conservative for discrete variables.

Table 8-4.
The KS Statistics for the Identified Features
from Chapter 7 for Dataset 1 and Dataset 2
Demonstrated that the Identified Features Were
Not from the Same Distribution

Kolmogorov-Smirnov Statistic for the Features Identified in Chapter 7		
Feature	***statistic***	***p-value***
URL Length	0.1622	0
<link type="text/css">	0.1455	0
server apache	0.1176	0
<a href> relative	0.116	0
Count of <a> tag	0.1148	0
Total HTML Tags	0.1118	0
<a href> OoD	0.1102	0
<iframe src="*..html">	0.1079	0
push()	0.1054	0
<img src="http*">	0.1012	0
<script async="true">	0.0967	1.37E-305
Total HTML Tags	0.0942	1.87E-289
<a href="https*">	0.089	1.84E-258
Count of <meta> tag	0.0859	5.84E-241
Count of <p> tag	0.0845	4.83E-233
<link href> OoD	0.084	2.10E-230
Count of 'l' character	0.0792	8.19E-205
Count of 'i' character	0.0611	3.33E-122
<a href="http*">	0.0525	1.76E-90
via 1.1	0.0421	2.18E-58
<form action="http*">	0.0372	1.58E-45
content-encoding gzip	0.0324	8.81E-35
vary user-agent	0.0316	3.82E-33
Count of 'p' character	0.0273	8.39E-25
Count of 'w' character	0.0242	1.26E-19
cache-control max-age	0.0242	1.58E-19
Count of 'z' character	0.0235	1.82E-18
url extension is .com	0.0217	8.33E-16
Total Extensions in URL	0.0204	5.52E-14
transfer-encoding chunked	0.0169	8.30E-10
Count of '.' character	0.0161	6.88E-09
Count of 'y' character	0.0065	0.0802
Count of 'f' character	0.0065	0.0839
Count of tag	0.0047	0.3592

The KS statistic, sometimes referred to as the D value, is the max distance between the two samples (the supremum). The null hypothesis stated that there was no difference between the two distributions. Thus, we can reject the null hypothesis if:

$$D_{n,m} > c(\alpha)\sqrt{\frac{n+m}{n \cdot m}}$$

and where:

α	0.1	0.05	0.025	0.01	0.005	0.001
$c(\alpha)$	1.22	1.36	1.48	1.63	1.73	1.95

Generally speaking, one can reject the null hypothesis $\alpha < 0.05$, which makes D = 0.007643 for our sample sizes m = 817,130 and n = 39,183. For all of the features except for the counts of the characters "y" and "f" and the count of the `<center>` element, we can reject the null hypothesis. Additionally, we observed small p-values except for the count of `<center>` elements, which is further evidence that we cannot reject the null hypothesis for this feature. This served as further evidence that Datasets 1 and 2 were not from the same population.

We then investigated the association between categorial features (those present in the HTTP header features) by calculating Pearson's chi square of association [242] and Cramer's phi [243] on the features from Datasets 1 and 2. Results are shown in Table 8-5.

Table 8-5.
Pearson's Chi Square and Cramer's Phi Showed that the Categorical Features Had Different Levels of Association with Maliciousness for Dataset 1 and Dataset 2

Association between respective features and maliciousness for HTTP columns for Dataset 1 / 2			
Features	***Pearson Chi-square***	***Cramer's phi***	***p-value***
`cache-control set max-age`	1466.8135 / 102.8165	0.1935 / 0.0109	0 / 0
`content-encoding gzip`	7416.1314 / 6.7207	0.4351 / 0.0028	0 / 0.0095
`server apache`	217.9999 /16.2768	0.0746 / 0.0043	0 / 0.0001
`transfer-encoding chunked`	3066.5364 / 92.4559	0.2798 / 0.0103	0 / 0
`vary user-agent`	20.1741 / 60.374	0.0227 / 0.0083	0 / 0
`via 1.1`	461.5291 / 159.1664	0.1085 / 0.0135	0 / 0

In Table 8-5, we first noticed large differences in Pearson's chi-squared value calculated on Dataset 1 compared to those calculated on Dataset 2, which suggested that the features of Dataset 1 had a stronger association with a website being malicious. We

also observed a difference in Cramer's phi (with 1 indicating total association and 0 indicating no association) between the respective datasets; specifically, the respective features for training and testing had a higher association in Dataset 1 than Dataset 2. This illustrated another difference between our two datasets. The p-value was low, indicating a significant result.

We observed that our best performing model from Chapter 7 was unable to accurately detect malicious websites in our new dataset. Upon further investigation, however, we observed various differences in the respective datasets that helped to explain this observation. However, for RQ8, we observed that we could not apply our best model to another dataset with success.

8.5.2 **RQ9: How do the Features Identified Perform on a New Dataset?**

Dataset 2 (the Alexa Top 1M with Cymon.io data) differed from Dataset 1 (the Alexa with Cisco Talos data) in several ways. First, Dataset 2 was much larger than Dataset 1 (approximately one million websites and approximately 47k websites, respectively). Secondly, the malicious websites from Dataset 2 were gathered from threat intelligence instead of from a security vendor. Thirdly, the two datasets were collected at different points in time. Thus, they were ultimately different datasets. In RQ8, we observed that we could not directly apply a detection model built in Chapter 7 to the Dataset 2. However, we still needed to investigate whether the features identified in Chapter 7 could successfully build detection models on this new dataset. With this research question, then, we explored how well the features from our prior models performed on a new (and different) dataset.

8.5.2.1 Retraining for Malicious Website Detection

We first explored building detection models on our second dataset, but with features identified in Chapter 7. We split our data into training and testing data and used the 34 features identified in Chapter 7, the 99 features gathered from prior research, and the 288 features achieved by dropping from our dataset those features that were consistent at least 95% of the time and by dropping from our dataset those features with high VIF values. Results are shown in Table 8-6 below.

Table 8-6.
Retraining on the New Dataset 2 Slightly Improved Detection Ability,
But Was Not Sufficient

Performance when Training a Random Forest Classifier on Dataset 2 with Features from Dataset 1							
Features	***FPR***	***FNR***	***ACC***	***AUC***	***MCC***	***Prec***	***Rec***
34 Identified in Ch7	0.0001	0.9905	0.9587	0.5046	0.0827	0.7741	0.0094
99 from Prior Research	0.0007	0.9704	0.9582	0.5143	0.1315	0.6419	0.0295
288 After First Feature Selection Step	0.0038	0.9117	0.9584	0.5422	0.1968	0.4972	0.0883

We saw slightly better results for the 34 features than for the random forest classifier in Table 8-1, with Table 8-6 showing an increase in MCC as we went from 34 features, to 99 features, and then to 288 features. However, we observed that we could not simply re-train our model on that new dataset "as is" and that considering additional features could be warranted.

8.5.2.2 Investigating Additional Features

In prior experiments, we noted the ability of the 34 features to detect malicious websites, though in the previous step we observed that the 34 features did not perform well even when we re-trained our models on the new dataset (though re-training did show improvement over using the model from Chapter 7 "as is"). It was possible, then, that additional features might improve our detection capabilities.

During our exploration, we identified two features that might show promise – the number of special URL characters and the number of "-" characters. To gain further assurance regarding the promise of those two features, we first measured the correlation (Pearson's correlation coefficient) of the respective feature with the target variable (whether the website was malicious). Full results are show in Table 8-7 below.

Table 8-7.
Pearson's Correlation Between Features
and Maliciousness in Dataset 2 Suggested
Ability of Two New Features for Detection

Correlation Values for Features in Dataset 2	
Features	***Correlation***
Count of '-' character	0.3660
Number of Special Chars in URL	0.2456
Count of 4-character words	0.0314
Total TLDs in URL	0.0303
URL extension is ".c"	0.0277
Count of 'a' character	0.0262
URL TLD "co" Count	0.0257
<link href="https*">	0.0252
<link rel=https://api.w.org/*>	0.0248
<link type="application/rsd+xml">	0.0245
<link rel="EditURI">	0.0245
<link rel="wlwmanifest">	0.0243
<link type="application/wlwmanifest+xml">	0.0243
<link rel="shortlink">	0.0243
<link rel="canonical">	0.024
<meta http-equiv="content-type">	0.0234
<meta http-equiv="Content-Type">	0.0234
<link rel="dns-prefetch">	0.0219
server nginx	0.0219
URL extension ".com"	0.0205
URL TLD "com"	0.0205
Count of 'u' character	0.0199
escape()	0.0196

In Table 8-7, we observed that the correlation value between these two new features was considerably higher than the rest of the features we had identified thus far. This suggested that we might want to consider using them.

8.5.2.3 Varying Ratios of Training to Testing Data

Since we had identified additional features that might improve our detection capability, we now rebuilt our RF model with these two features. Additionally, we varied the train-to-test ratio (see Table 8-8 below). We tuned the model parameters with an F1 scoring metric.

Table 8-8.
Incorporating Two Additional Features Greatly Improved Detection Ability

Performance when Retraining a Random Forest Classifier on Dataset 2 with Identified +2 Features with Various Training: Testing Ratios							
Training : Testing	***FPR***	***FNR***	***Acc***	***AUC***	***MCC***	***Prec***	***Rec***
70% : 30%	0.0772	0.2151	0.9174	0.8549	0.4591	0.3067	0.7868
60% : 40%	0.0774	0.2127	0.9169	0.8537	0.4574	0.3058	0.7848
50% : 50%	0.0774	0.2116	0.9146	0.8521	0.4512	0.2991	0.7839
40% : 60%	0.0778	0.2105	0.9170	0.8554	0.4589	0.3061	0.7883
30% : 70%	0.0620	0.2724	0.9145	0.8518	0.4508	0.3052	0.7894
20% : 80%	0.0783	0.2128	0.9160	0.8544	0.4562	0.3036	0.7871
10% : 90%	0.0791	0.2121	0.9153	0.8543	0.4544	0.3012	0.7878

After incorporating the two additional features, we saw a large performance increase that was consistent across Dataset 2. From this observation, we postulated that the features from Chapter 7 remained relevant, though some slight modifications would need to be made in order to improve malicious website detection.

8.5.2.4 Identifying Training to Testing Ratio

We had observed that the addition of the two features – the number of special characters and the number of "-" characters – improved malicious website detection. We also observed that the MCC remained fairly consistent as we varied the training-to-testing

ratio. As a result, we further investigated how much training data was actually needed to build the models thus far. Results are shown below in Table 8-9. We tuned the model parameters and also performed grid search on the `class weight` parameter of the `Scikit`-Learn [29]. In Table 8-9 below, we report that we received consistent results even when we used just 3% of the data for training.

Table 8-9.
Detection Performance When Incorporating Two Additional Features Remained Consistent with 3% of Data Used for Training

Performance when Retraining and Tuning a Random Forest Classifier on Dataset 2 with Identified +2 Features with Lower Training Ratios								
Train: Test Split	***FPR***	***FNR***	***ACC***	***AUC***	***MCC***	***Prec***	***Rec***	***Grid Search of Class Weight***
0.05:0.95	0.0796	0.2160	0.9146	0.8521	0.4512	0.2991	0.7839	No
0.03:0.97	0.0797	0.2165	0.9145	0.8518	0.4508	0.2989	0.7834	No
0.03:0.97	0.0747	0.2312	0.9187	0.8470	0.4548	0.3087	0.7687	Yes

In RQ9, we observed how well the features identified in Chapter 7 performed on a new dataset. Alone, and even with re-training, the 34 features did not demonstrate the ability to detect malicious websites. Upon further investigation, however, we identified two additional features that greatly complemented the detection ability of the 34 identified features. As such, we observed that we could reuse the features from our previous studies, though we also needed to investigate potential additions.

8.5.3 RQ10: What Aspects from Prior Experiments Can We Apply to Our New Dataset?

In RQ8, we observed that our best performing models from Chapter 7 did not perform well on the new dataset. However, we did observe differences in the respective training and evaluation datasets. With RQ10, we investigated the impact of using aspects of both models on detection capability.

8.5.3.1 Training Dataset Evaluation

For the first step, we trained the models with both Dataset 1 and Dataset 2 and evaluated the models using Dataset 2. Results appear in Table 8-10 below.

Table 8-10.
Incorporating Dataset 2 Into Training Did Not Improve Detection Ability
on Dataset 2 When Using Identified Features

Performance when Training a Random Forest Classifier with Identified on Dataset 1 and 2 and Evaluating on Dataset 2 with 34 Identified Features							
Fraction of Dataset 1: Fraction of Dataset 2	***FPR***	***FNR***	***ACC***	***AUC***	***MCC***	***Prec***	***Rec***
0.8 Dataset 1: 0.01 Dataset2	0.6536	0.3917	0.3572	0.4773	-0.0189	0.0387	0.6082
0.8 Dataset 1: 0.05 Dataset2	0.7293	0.3129	0.2879	0.4788	-0.0189	0.0392	0.6870
0.8 Dataset 1: 0.1 Dataset2	0.7094	0.3350	0.3060	0.4777	-0.0195	0.0390	0.6649
0.8 Dataset 1: 0.2 Dataset2	0.7221	0.3196	0.2946	0.4791	-0.0185	0.0392	0.6803
0.8 Dataset 1: 0.3 Dataset2	0.7366	0.2991	0.2815	0.4820	-0.0162	0.0395	0.7008
0.8 Dataset 1: 0.4 Dataset2	0.7138	0.3204	0.3024	0.4828	-0.0151	0.0396	0.6795
0.8 Dataset 1: 0.5 Dataset2	0.6357	0.4109	0.3735	0.4766	-0.0193	0.0386	0.5890

In Table 8-10, we see that models trained with Datasets 1 and 2 were unsuccessful at detecting malicious websites in Dataset 2. Therefore, we investigated whether we could incorporate new data into the training of our models to detect threats from Dataset 1 as well as Dataset 2. We did this on the set of 34 features as well as on the set of 99 features gathered from prior research. We also varied the train to test split by training on 20%, 30%,..,70% and evaluating on 80%, 70%,..,30% respectively.

Table 8-11.
Training Models with Both Dataset 1 and Dataset 2 Slightly Improved Detection
on Both Datasets When Using Identified Features

Performance with Identified Features When Training Using Dataset 1 and 2 and Testing on Dataset 1 and 2							
Train / Test Split	***FPR***	***FNR***	***ACC***	***AUC***	***MCC***	***Prec***	***Rec***
20:80	0.0014	0.9211	0.9574	0.5387	0.2292	0.7199	0.0788
30:70	0.0013	0.9183	0.9576	0.5401	0.2368	0.73967	0.0816
40:60	0.0013	0.9162	0.9576	0.5411	0.2403	0.74231	0.0837
50:50	0.0013	0.9121	0.9580	0.5432	0.2468	0.74524	0.0878
60:40	0.0014	0.9114	0.9583	0.5435	0.2461	0.7360	0.0885
70:30	0.0015	0.9100	0.9583	0.5442	0.2476	0.7339	0.0899

Table 8-12.
Training Models with Both Dataset 1 and Dataset 2 Slightly Improved Detection on Both Datasets When Using Features from Prior Research

Performance when Training a Random Forest Classifier with Prior Features Using Dataset 1 and 2 and Testing on Dataset 1 and 2							
Train / Test Split	***FPR***	***FNR***	***ACC***	***AUC***	***MCC***	***Prec***	***Rec***
20:80	0.0008	0.9190	0.9578	0.5401	0.2489	0.8153	0.0809
30:70	0.0008	0.9160	0.9579	0.5415	0.2537	0.8172	0.0839
40:60	0.0009	0.9082	0.9580	0.5454	0.2647	0.8131	0.0917
50:50	0.0010	0.9072	0.9579	0.5458	0.2645	0.8047	0.0927
60:40	0.0012	0.9002	0.9582	0.5492	0.2717	0.7908	0.0997
70:30	0.0012	0.9010	0.9582	0.5488	0.2709	0.7917	0.0989

We observed that with training, we could slightly improve our detection ability when training and evaluating on both datasets. However, we noted a very high FNR, which implied that this technique, despite producing a high accuracy, was not feasible. We further investigated the impact of over-sampling with two separate techniques: SMOTE [186] (provided by [187]) and adaptive synthetic sampling (ADASYN) [244].

Table 8-13.
Over-Sampling Slightly Decreased Detection Performance When Training Models with Both Dataset 1 and Dataset 2 and Evaluating on Dataset 1 and Dataset 2 with Identified Features

Performance when Training a Random Forest Classifier with Over-sampling on Dataset 1 and 2 with Identified Features and Evaluating on Dataset 1 and 2							
Over-sampling method	***FPR***	***FNR***	***ACC***	***AUC***	***MCC***	***Prec***	***Rec***
SMOTE	0.0048	0.9074	0.9547	0.5438	0.1947	0.4730	0.0925
ADASYN	0.0037	0.9120	0.9556	0.54212	0.2018	0.5247	0.0879

Table 8-14.
Over-Sampling Slightly Decreased Detection Performance When Training Models with Both Dataset 1 and Dataset 2 and Evaluating on Dataset 1 and Dataset 2 with Prior Features

Performance when Training a Random Forest Classifier with Over-sampling on Dataset 1 and 2 with Prior Features and Evaluating on Dataset 1 and 2							
Over-sampling method	***FPR***	***FNR***	***ACC***	***AUC***	***MCC***	***Prec***	***Rec***
SMOTE	0.0066	0.8229	0.9566	0.5852	0.2976	0.5573	0.1770
ADASYN	0.0081	0.8013	0.9556	0.5952	0.3079	0.\5344	0.1986

We observed slight average improvement as well as high FNRs when using the prior features versus the features we identified.

8.5.4 **Discussion**

We first observed that even our best model performed poorly when applied "as-is" to another dataset. However, there were differences in the dataset, particularly in the URL features that were identified by t-SNE plots. This result was not surprising, given that prior authors had often received high error rates when evaluating their models on different datasets without any re-training. Additionally, we found that there were several differences in the respective features' correlation and association to maliciousness between the datasets. We also observed that the features from the respective datasets did not come from the same distribution.

Secondly, we found that the 34 features we identified in Chapter 7 and the features gathered from prior research demonstrated potential for detection on a new dataset, however new features needed to be incorporated to make the detection models successful. Specifically, we explored the potential of other features via correlation, which motivated their incorporation into a detection mechanism. Once we incorporated these features, we saw large improvement in our detection ability.

Thirdly, we observed that even when we used both Dataset 1 and Dataset 2 for training, we were still unable to detect malicious websites in Dataset 2. However, the models trained from Datasets 1 and 2 were better able to detect malicious websites from Datasets 1 and 2, but missed a substantial portion of websites, as demonstrated by the high FNR.

8.6 **Conclusion**

In this chapter, we detailed our investigation through three research questions that explored the application of models trained on one dataset to another dataset. We first

observed that a model trained on another dataset could not be applied “as-is” to another dataset with guaranteed success. This result reflected findings of prior authors, who had often received high error rates when evaluating their models on different datasets without any retraining.

Secondly, we found that the 34 features identified in Chapter 7 demonstrated slight potential on a new dataset and served as a good foundation for features, though modifications were required. Specifically, these features could be reused, but other features needed to be incorporated based on the dataset. Once we incorporated two new features derived from Dataset 2, we observed improvement in our detection ability.

Thirdly, we observed that even when we used both Dataset 1 and Dataset 2 for training, we still were unable to detect malicious websites in Dataset 2. While the models trained from Datasets 1 and 2 showed slight improvement, we still missed a substantial portion of malicious websites.

Chapter 9: A Temporal Evaluation of Feature-Based Malicious Website Detection

9.1 Introduction

Web security is a fast-moving field. Attackers and defenders are constantly creating new techniques to confront each other [221] and detecting malicious websites, used by attackers for phishing, drive-by downloads, and C2, is a challenge. The adversarial environment of web security and relationship between offensive and defensive practitioners motivates research and industry to continuously explore new techniques and tools. Defenders in research and industry have used features and machine learning to detect malicious websites yielding promising results. However, several studies including [42], [50], [64], [75], and [159] have observed that detection models do not remain robust over time. Other studies like [91] have shown success when training and testing on data gathered across different points in time. There is a lack of clarity regarding the ability of detection models to remain robust over time. Given that attacks change over time, there is an inherent assumption that detection models, especially those built with either supervised or unsupervised learning on current or past data, will eventually become inaccurate. This assumption however, has been minimally explored.

In this chapter, we perform a temporal evaluation of feature-based malicious website detection. We study 106,766 websites from the Alexa Top 1M [112] provided by [176] over a period of 12 weeks. We use Google Safe Browsing [132] to label the websites as malicious or benign. We build detection models with the random forest algorithm and three sets of features gathered from a `response` to web `request` – 34 identified in Chapter 7 from a dataset (Dataset 1) provided by Cisco Talos Intelligence Group [177], 99 gathered from prior research, and 41 re-selected from the dataset used in

this study (Dataset 3). We analyze the impact of re-training and measure the change in websites and detection performance over time. Overall, we observe that 1) detection models slowly degrade over time with an exponential decay however improve to a power decay with re-training, and 2) websites (as defined by their features) change more as time grows.

We make the following contributions:

- We present, to our knowledge, the first study of feature-based malicious website detection that focuses on detection performance and change over time;
- We demonstrate that while retraining detection models improves performance and can result in a slower performance degradation, performance still decreases over time; and
- We present a new method of analyzing and measuring change in website datasets which enables further statistical analysis.

9.2 Related Research

Related research in studying websites over time falls into two categories - research focused on if and how the internet and webpages change over time, and research into malicious website detection that includes temporal aspects (a model trained at one point in time and applied on data from a later point in time).

Researchers in the first group focused on examining the dynamic nature of the web. Websites change with some studies having quantified and measured this change. Web crawlers, which iterate webpages and the internet, are useful in studying website changes. For example, [168] used a crawler to determine that 40% of all webpages in

their dataset were subject to change (based on the MD5sum). The MD5sum determines whether a sequence of bytes (in the form of a webpage [168]) changes. Features like the webpage length and HTTP `response` code can also be used to determine change. For example, [168] monitored changes in the website. Researchers [173] and [174] leveraged additional features including word level and DOM-related features to characterize website changes and HTML element persistence. HTTP status codes have been combined with approaches from [170] and are used to determine the similarity of webpages to each other, as in [169]. In [170] Fetterly expanded on [171] and observed that 40% of webpages in their dataset changed within a week. Authors [172] aimed to infer change rates of webpages on the web. Researchers in [175] proposed criteria and a new metric to measure website change though this metric was not presented in the context of malicious website detection.

While studying detection performance over time has not been the primary focus in malicious website detection research, a few works have evaluated their detection methods on the same dataset at a later point in time. Zarras et al. [82] evaluated Bothound on data collected over time. On the first evaluation, their technique identified 718 domains as malicious, 74.7% of which were found in denylists. On the second dataset, collected one week later, they found that an additional 59 identified domains (for a total of 82.9%) were now on denylists. Their approach specifically identified malicious domains generated by malware. Basnet et al. [64] observed over a 900% increase in the error rate (from 0.42% to 3.82%) when training and testing on dataset separated by three months and investigated different training frequencies. They concluded that models must be re-trained to adjust to changing phishing tactics.

Other studies have not shown much performance decrease when training and testing models on data collected at different times however, in such studies, the difference between training and evaluation is closer than in the studies that show a larger performance decrease. Prophiler [45] achieved an FPR of 9.88% and an FNR of 0.77%, but did so on a validation dataset collected immediately following the training dataset. Some researchers have incorporated temporal aspects by evaluating live feeds of data. Ma et al. in [36]-[37] ingested live feeds of data from a Webmail provider containing samples of spam and phishing URLs, and leveraged online learning to investigate different training regimens. They showed the benefits of continuous training and observed a decrease in the cumulative error rate from approximately 2% to 1% over a 100-day period. CANTINA+ [46] observed a 92.25% true positive and a 1.375% false positive rate when training on a dataset and evaluating on another dataset two weeks later. Whittaker et al [42] were able to achieve a phishing detection true positive rate of 91.85% and a false negative rate of 0.01% when training on three months of data and evaluating on another dataset two weeks later. Marchal et al. [91] achieved highly accurate results (an AUC of 0.999) on phishing webpage detection and trained and tested on datasets gathered one month apart.

In this chapter we focus on observing and measuring malicious website detection performance and change over time. Like others including [36]-[37], [42], [46], [64], and [91] we perform analysis of detection models that were trained and evaluated on datasets gathered at different times. Also like [36]-[37] and [64], we investigate different training frequencies. The differentiators in our study are that we 1) make the performance change over time our main focus, 2) substantiate our observations regarding the performance

change with measurable rationale, and 3) limit our analysis to the same dataset gathered over time.

9.3 Research Questions

9.3.1 Research Question 11

This research question focuses on the investigation of the performance of detection models over time. In Chapter 7, we were able to build several detection models, with our best performing model achieving an MCC of 0.9174 with features we identified through feature selection. However, we had little insight into how these models would perform over time and did not have insight into their consistency when applied on a new dataset. In the portion of our research outlined in Chapter 8, we demonstrated that re-training and adjusting models was needed when applying models built from one dataset to another. Specifically, models trained on one dataset could not necessarily be applied to another. Prior researchers have observed different results. Some have seen performance decrease as in [64] when their training and testing sets were collected at different times and some, like [91], have been able to achieve high detection metrics when separating their training and testing set by a few weeks. Insight into if and how detection models change over time may influence if and when a researcher or practioner decides to re-evaluate or re-train their detection models. The differences of results presented in prior research and the knowledge gained from studying performance over time leads us to the next research question. RQ11, then, is stated as follows:

RQ11: How does detection performance change over time?

9.3.2 **Research Question 12**

To understand whether detection models can remain robust over time (and to determine the potential reasons they remain robust or fail to do so), we first must understand whether websites change over time. Based on prior research, we hypothesized that websites change over time and that malicious website detection models will eventually become irrelevant and no longer be able to distinguish between benign and malicious websites, though we had not yet established this in our research. To do so, we determined whether the features that compose a website (and are used for detection) change over time. Gaining insight into feature change (and whether they are capable of detecting malicious websites or were related to features that have demonstrated the ability to detect malicious websites) was a necessary step for constructing models that remain relevant over time. RQ12 is stated as follows:

RQ12: Do websites change over time?

9.3.3 **Research Question 13**

We extended RQ12 further in our final research question by evaluating website change more thoroughly by examining feature change over time. Specifically, we gathered several data points regarding the number of features that changed when we compared the time between data collections. Access to 12 weeks of data enabled us to perform various comparisons (comparing the snapshot of week one to week two, week one to week three, and week one to week four, then comparing the snapshot of week two to week three, week two to week four, etc.). RQ13 is stated as follows:

RQ13: If websites change over time, how much do they change over time?

9.4 Approach

Our approach is comprised of three steps. In the first step we collected our dataset (Dataset 3) over a period of 12 weeks that was derived from the Alexa Top 1M and Google Safe Browsing. In the second step we selected feature sets to build detection models – 34 features identified in Chapter 7, 99 features from prior research (also used in Chapter 7), and a set of features re-selected from Dataset 3. In the final step we build and evaluate detection models across snapshots and compare the websites (and their features) from the respective snapshots to each other. The process is depicted in Figure 9-1 below.

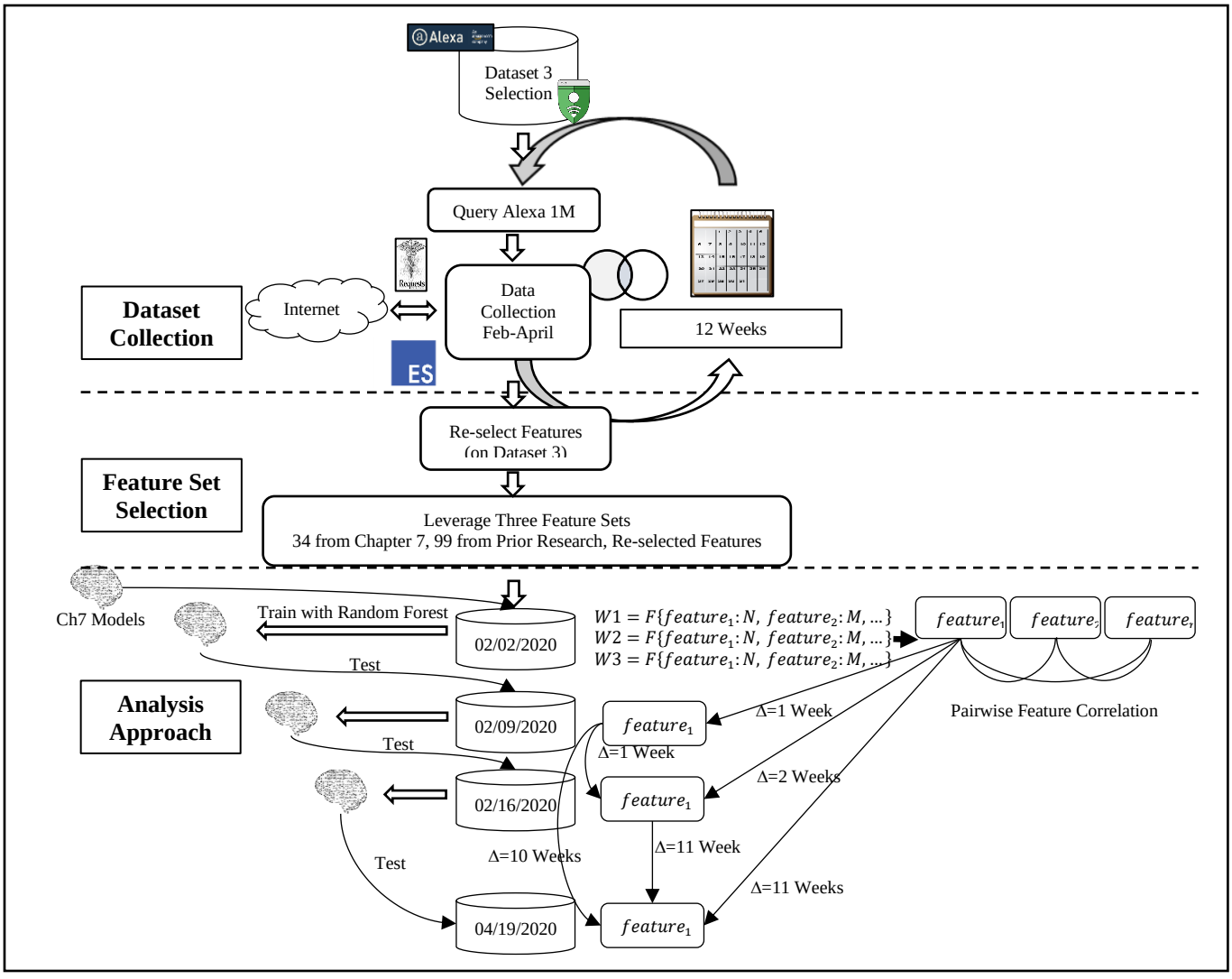


Fig. 9-1. Three step approach for temporal evaluation of feature-based malicious website detection (Images courtesy of Pixabay [22].)

9.4.1 **Dataset Collection**

We derived the dataset from the websites that were consistently present in the Alexa Top 1M over a period of 12 weeks. The choice of a period of 12 weeks was derived from prior studies. Fetterly collected webpages over a 10- and 11-week period in [169] and [170], respectively. Basnet [64] observed a 900% increase in error rate when using training and testing datasets separated by three months. Beginning on February 2nd, 2020, we performed a query via Censys for the Alexa Top 1M. Over the following week, we gathered features from the websites by performing an HTTP `GET request` to each website. On each of the following weeks for a total of 12 weeks, we re-performed the query for up-to-date Alexa Top 1M data and re-performed our gathering of data from the respective websites. We limited our analysis to websites that were present in the Alexa Top 1M during all 12 weeks. Hence, our final dataset consisted of snapshots of 106,766 websites that remained consistent over the 12-week period. Table 9-1 shows the number of websites on the Alexa Top 1M that were consistent over the respective time periods.

Table 9-1.
A Fraction of the Websites in the Alexa
Top 1M Were Consistent Over Time

Number of Consistent Websites in Alexa Top 1M	
Start Date - End Date	***Common Websites***
2/2/2020-2/9/2020	364,106
2/2/2020-2/16/2020	281,515
2/2/2020-2/23/2020	238,459
2/2/2020-3/1/2020	203,145
2/2/2020-3/8/2020	185,673
2/2/2020-3/15/2020	164,776
2/2/2020-3/22/2020	144,578
2/2/2020-3/29/2020	129,295
2/2/2020-4/5/2020	118,418
2/2/2020-4/12/2020	111,618
2/2/2020-4/19/2020	106,766

Ground truth data (the designation of which websites were malicious and which were benign) consisted of data gathered from Google Safe Browsing. We labeled this dataset (106,776 websites labeled with Google Safe Browsing) as Dataset 3.

9.4.2 Feature Set Selection

We narrowed our focus to the 34 features identified in Chapter 7 and to the 99 features gathered from prior research and also used in Chapter 7. The 34 features have been identified in our studies (Chapter 7) as being able to detect malicious websites with an MCC of up to 0.9281 on a prior dataset. The 99 features have been vetted throughout prior research. Additionally, we re-performed the feature selection process from Chapter 7 on the new dataset in order to arrive at a third set of features for analysis in case the 34 features and 99 features were not effective on this new dataset. We used `Esprima` [245] to parse the content of the HTTP `response`.

9.4.3 Analysis Approach

Our analysis was divided into three sections corresponding to the three proposed research questions. The process is detailed below.

Step 1: Investigate model performance over time

- Evaluate prior model performance
    - Train an RF [98] model using Dataset 1 (from Chapters 4–7), using the 34 features identified in Chapter 7 and the 99 features gathered from prior research;
    - Evaluate the performance of the RF model trained on Dataset 1 relative to the respective snapshots (collection of benign and malicious websites) of Dataset 3 – the 106,766 websites, that are consistent across the 12 weeks – and record the result;
    - Identify the following:
        - The number of website detection outcomes that are consistent throughout the 12 weeks,
        - The number of website detection results that changed classification, and
        - The accuracy of the detection results based on the ground truth data;
- Retrain and evaluate a new RF model
    - Retrain an RF detection model on the first snapshot of Dataset 3 (the Alexa Top 1M that are consistent over the 12 weeks beginning February 2, 2020) with Google Safe Browsing as ground truth;
    - Evaluate the performance of the model trained on the first snapshot compared to the remaining 11 snapshots and evaluate the

performance including whether detection performance increases, decreases, or remains constant over time;

- Retrain an RF model on each snapshot of Dataset 3 (and evaluate it on the later snapshots) to determine the following:
  - Whether performance increases or decreases, and
  - How the performance compares to the model trained on the first snapshot of data and evaluated based on the proceeding snapshots;
- Retrain an RF model on all previous snapshots of Dataset 3 and evaluate on all proceeding snapshots to determine:
  - Whether performance increases or decreases, and
  - How the performance of the model trained on the first snapshot of data and evaluated on the proceeding snapshots compares to the performance of the model trained on all previous snapshots and evaluated on the proceeding snapshots.

Step 2: Determine whether websites (composed of features) change over time

We hypothesized that we would see some change in performance over time and we performed this step to gain insight into potential reasons. For this step, we evaluated each feature in isolation to determine whether it changed. We defined a website as a set of key-value pairs where the key is the feature and the value is the respective quantification of that feature. As such we defined a website as the following:

$$W = F\{feature_1 : N, feature_2 : M, \ldots . feature_t : Z\}$$

where W is the website, F is a set of key-value pairs, $feature_1$, $feature_2$, …$feature_t$ are the features, and N, M, and Z are integers (the respective values of each feature).

With this definition of a website, we derived a histogram from the values of a specific feature in a snapshot. Figure 9-2 below provides a contrived example (not based on real data and only used for demonstrative purposes) of a feature in a specific snapshot – the number of HTML tags - which is a feature we collected in this study. This is one example and we applied this approach to the other features in each weekly snapshot.

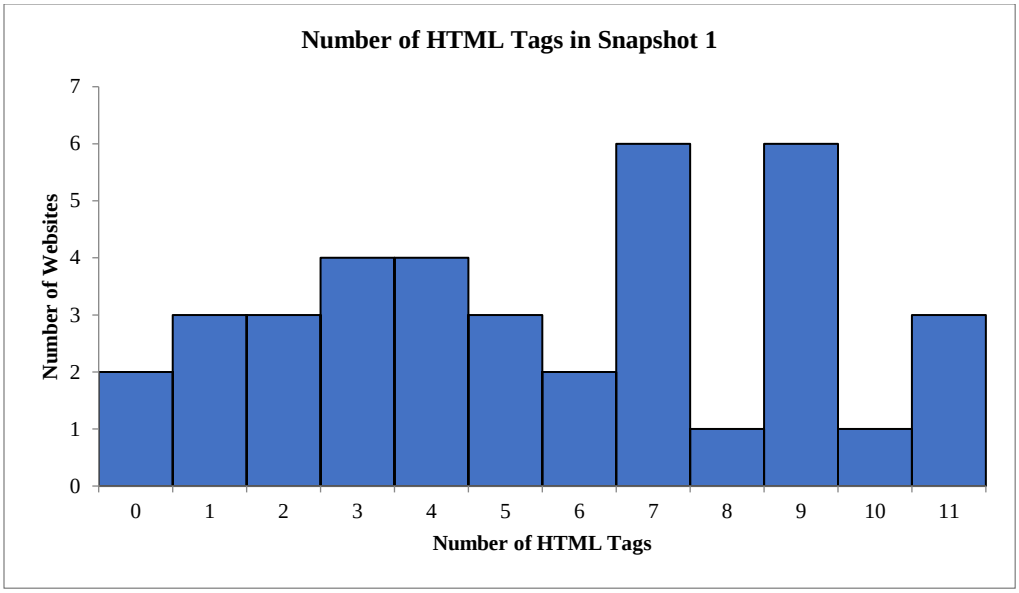


Fig. 9-2. Distribution of the number of HTML tags

We derived a histogram from the collection of measurements for each feature in each snapshot. The x axis represents the value of the feature and the y axis represents how many websites have the specific of that feature. The histogram in Figure 9-2 can be read as; two websites have a zero for the value of the "Number of HTML Tags" feature, three websites have a value of one for the number of HTML tags,…, and three websites have a value of 11 for the number of HTML tags. This histogram was created for each feature in each snapshot. This view of the data formed the basis for the statistical tests

used to determine which features changed and which did not change from snapshot to snapshot. We leveraged one strategy incorporated by industry and four statistical tests. First, we investigated the pairwise correlation between each feature in the respective snapshots, a strategy frequently employed by industry for determining whether a model should be re-trained. Next, we used four statistical tests – the t-test for related samples [194]-[195], the two-sample KS test [196]-[197], the k-sample Anderson-Darling test [198]-[199] (where k =2), and the Kruskal Wallis H test [200]-[201]. The related t-test for the null hypothesis determined whether the two related or repeated samples had identical average values, relied on dependent observations, and assumed normality. However, since our sample size is large (greater than 31) we rely on the Central Limit Theorem [246] should any data be non-normal. Furthermore, [247] validates the reliance on the Central Limit Theorem for large (and potentially non-normal) datasets. The two-sample KS tested that two independent samples were drawn from the same distribution and required independent observations. The k-sample Anderson-Darling [198]-[199] tested that k-samples were drawn from the same population and required independent observations. The Kruskal Wallis H test [200]-[201] tested whether the population median of all the groups was equal (the test is a non-parametric version of ANOVA [248]) and also required independent observations. We considered two possibilities of the samples in our dataset – the samples are dependent (or related) and the samples are independent. An argument can be made for both cases. In the case of dependent observations, the most appropriate test was the related t-test, since the samples could be considered related (the same websites collected over time). However, given the dynamic nature of the internet and the novelty of studying feature change in this manner, we also

adopted the view that websites collected at different times could be considered independent, or not related. Hence, we also used the other three tests – the two-sample KS test [196]-[197], the k-sample Anderson-Darling [198]-[199], and the Kruskal Wallis H test [200]-[201] – and observed the outcomes.

9.5 Results

9.5.1 RQ11: How does Detection Performance Change Over Time?

For this research question, we applied to Dataset 3 the RF model built from Dataset 1 with 34 features identified in Chapter 7 and the RF model built from Dataset 1 with 99 features gathered from prior research. Tables 9-2 and 9-3 below present a summary of the number of websites that were consistent and the number of websites that changed.

Table 9-2.
The Detection Model Built from
Dataset 1 with 34 Features
Remained Consistent on Dataset 3

Website Prediction Based on Mode Trained on Dataset 1 and 34 Identified Features		
Consistency Status	***Number***	***Percent***
Consistent Benign	89257	83.6
Consistent Malicious	438	0.4
Failed Collection	7794	7.3
Changes	9277	8.7

Table 9-3.
The Detection Model Built from
Dataset 1 with 99 Features
Remained Consistent on Dataset 3

Website Prediction Based on Mode Trained on Dataset 1 and 99 Prior Features		
Consistency Status	***Number***	***Percent***
Consistent Benign	89204	83.5
Consistent Malicious	491	0.5
Failed Collection	7794	7.3
Changes	9277	8.7

Tables 9-2 and 9-3 show the metrics of classification consistency from the models built from the 34 features and 99 features from Dataset 1 and applied to Dataset 3. In both cases approximately 84% of the websites were consistently classified as benign; 0.4% of the websites were consistently classified as malicious; 7% had at least one failure during collection; and 9% changed their classification over the 12 weeks. Although we had 85% of the websites with no collection failures (the connection timed out, the connection was blocked, etc) that were consistent with respect to their classification by the models built in Chapter 7, we did not yet know how well those the respective models performed (if they were accurate). To determine this, we captured performance metrics and used Google Safe Browsing as the ground truth. Results are shown in Tables 9-4 and 9-5.

Table 9-4.
The Model Trained on Dataset 1 with 34 Features
Performed Consistently Poorly When Applied to Dataset 3

Performance of Random Forest Classifier Trained On 34 Identified Features on Dataset 1 Applied to Dataset 3 Over Time						
Week	***FPR***	***FNR***	***AUC***	***MCC***	***Prec***	***Rec***
2/2/2020	0.0044	0.9822	0.5066	0.0311	0.0989	0.0177
2/9/2020	0.0045	0.9829	0.5062	0.0289	0.0934	0.0170
2/16/2020	0.0045	0.9833	0.5060	0.0279	0.0907	0.0166
2/23/2020	0.0046	0.9830	0.5061	0.0281	0.0909	0.0169
3/1/2020	0.0045	0.9829	0.5062	0.0291	0.0938	0.0170
3/8/2020	0.0045	0.9833	0.5060	0.0281	0.0918	0.0166
3/15/2020	0.0045	0.9829	0.5062	0.0288	0.0928	0.0170
3/22/2020	0.0045	0.9826	0.5063	0.0295	0.0949	0.0173
3/29/2020	0.0046	0.9831	0.5060	0.0279	0.0907	0.0168
4/5/2020	0.0046	0.9823	0.5065	0.0299	0.0952	0.0176
4/12/2020	0.0047	0.9831	0.5060	0.0276	0.0896	0.0168
4/19/2020	0.0046	0.9838	0.5057	0.0267	0.0884	0.0161

Table 9-5.
The Model Trained on Dataset 1 with 99 Features
Performed Consistently Poorly When Applied to Dataset 3

Performance of Random Forest Classifier Trained On 99 Prior Features on Dataset 1 Applied to Dataset 3 Over Time						
Week	***FPR***	***FNR***	***AUC***	***MCC***	***Prec***	***Rec***
2/2/2020	0.0035	0.9837	0.5063	0.0326	0.1108	0.0162
2/9/2020	0.0036	0.9841	0.5061	0.0313	0.1067	0.0158
2/16/2020	0.0036	0.9840	0.5061	0.0316	0.1076	0.0159
2/23/2020	0.0036	0.9837	0.5062	0.0322	0.1090	0.0162
3/1/2020	0.0035	0.9840	0.5061	0.0317	0.1081	0.0159
3/8/2020	0.0035	0.9845	0.5059	0.0310	0.1069	0.0154
3/15/2020	0.0035	0.9845	0.5059	0.0307	0.1055	0.0154
3/22/2020	0.0036	0.9842	0.5060	0.0313	0.1073	0.0157
3/29/2020	0.0036	0.9846	0.5058	0.0298	0.1030	0.0153
4/5/2020	0.0036	0.9838	0.5062	0.0318	0.1079	0.0161
4/12/2020	0.0037	0.9846	0.5058	0.0297	0.1025	0.0153
4/19/2020	0.0036	0.9849	0.5056	0.0292	0.1018	0.0150

In both cases, the models built on the 34 identified features and 99 prior features performed only slightly better than random on each snapshot. (We saw an MCC of 0.0311 for the model built with the 34 features, and an MCC of 0.0326 for the model built with the 99 features.) We went on to investigate the performance of models over time, our primary goal being to gauge how long models would remain accurate for detection before becoming out-of-date. To that end, we trained an RF detection model on the first week of collection and evaluated its performance on the remaining 11 weeks of collection. We used Google Safe Browsing as ground truth data for evaluating the

accuracy. For re-training, we used the 34 identified features from Chapter 7 derived from Dataset 1, the 99 features gathered from prior research, and 41 re-selected features from Dataset 3. We performed re-selection on Dataset 3 in case there was another set of features that better suited Dataset 3, following the same process outlined in Chapter 7 for feature selection. Figure 9-3 below shows the performance of the models trained on the first week of collection and applied to the 11 subsequent weeks. We focused on the MCC and FNR. The FPR is not shown because it only changed slightly and was very low (approximately equal to or less than 0.01% over all measurements). Full results can be found in Appendix D.

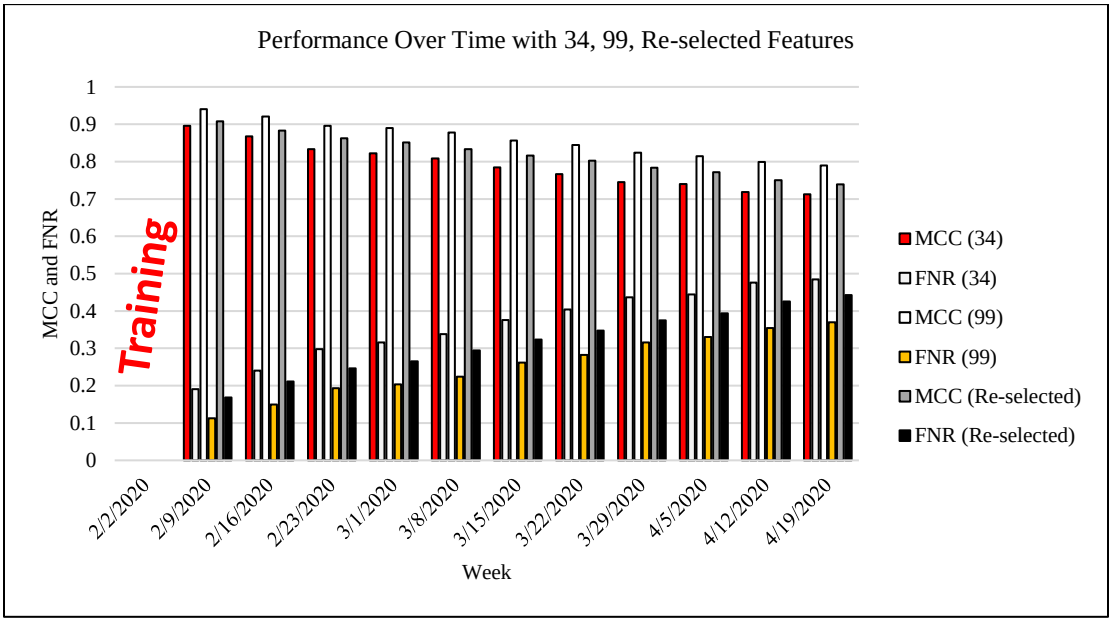


Fig. 9-3. Performance consistently decreased when training on the first snapshot of Dataset 3 and evaluating on future snapshots

In all three cases, we observed a decrease in MCC over the 12 weeks; over the same period, we observed an increase in the FNR with all three sets of features. We also observed that the model built from 99 features performed slightly better over time than the model built with 34 features, although both models exhibited similar behavior with

respect to performance degradation over time. We observed that the model built from re-selected features outperformed the model built from 34 features from Chapter 7, but it underperformed when compared to the model built from the 99 features gathered from prior research. Additionally, we observed that the performance (MCC) of models built from all three feature sets decreased with an exponential decay $N = N_0e^{-\lambda t}$ which we determined by performing linear and non-linear regressions on the sequence of data finding the regression with the smallest error. On the linear regressions, we observed R^2 values 0.9803, 0.9918 and 0.9961, for the 34, 99, and re-selected features with p-values of 5.43E-9, 1.04E-10, and 3.56E-12, respectively.

We then examined whether and how re-training could improve the ability to distinguish between benign and malicious websites. To do so, we re-trained on each week and evaluated on all subsequent weeks. The results were similar when re-training on the different snapshots. See Figure 9-4 for the results of re-training on the sixth week. Full results appear in Appendix D.

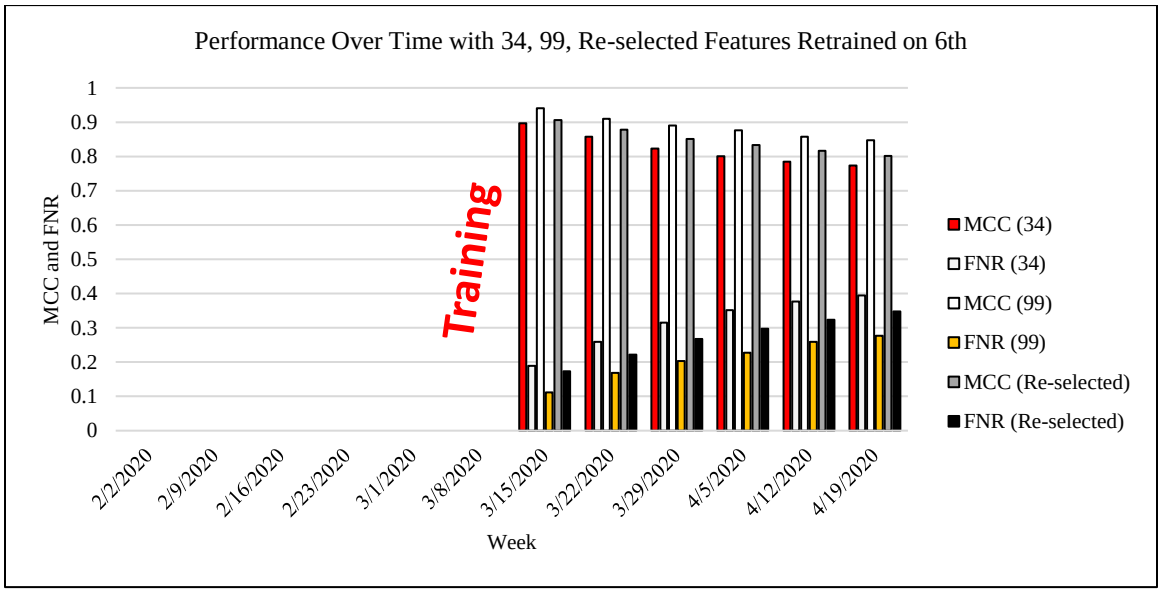


Fig. 9-4. Performance temporarily increased when retraining, but still consistently decreased over time

When re-training on the sixth snapshot (taken March 8th, 2020), we observed a performance increase when applying the re-trained models (with 34, 99, and re-selected features, respectively) to the following weeks, followed by a decrease similar to the model performance from Figure 9-3. After re-training, we also observed that the performance for all three feature sets decreased according to a power rule $N = N_0 t^{-n}$ based on the result of performing non-linear and linear regressions. On the linear regressions, we observed R^2 values 0.9499, 0.9719 and 0.9825, for the 34, 99, and re-selected features with p-values of 9.5E-4, 2.9E-4, and 1.1E-4, respectively. Additionally, the MCC and FNRs of models trained on the snapshot taken the week of March 8th, 2020 and evaluated on the data from weeks 6, 7, …11, was approximately equal to the MCC and FNR of the model trained on February 2nd, 2020, and applied to the February 9th, 2020, snapshot (Figure 9-3).

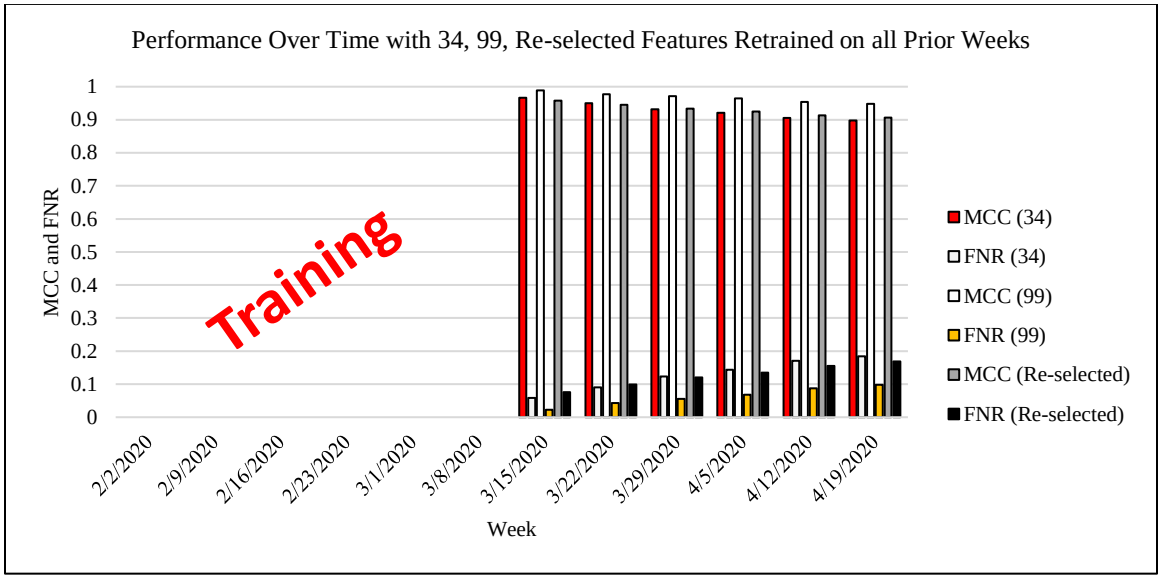


Fig. 9-5. Model performance improved and remained more robust when training on several snapshots of prior data

We then evaluated the impact of using all of the previous weeks as training data instead of using just the previous week. Results are shown in Figure 9-5, with full results appearing in Appendix D. When doing so, we noted improvement in the performance (MCC and FNR). We observed a decrease in performance on the subsequent weeks, though the decrease was slower than that for the models that were trained on a single snapshot of data. Again, the model built from 99 features performed the best and the re-selected features outperformed the 34 features identified in Chapter 7, though the difference between the three was smaller (see Figure 9-5 versus Figures 9-4 and 9-3). Figure 9-5 displays the observed power decrease for the 34 features and the re-selected features and there is an observed exponential decrease for the 99 features. On the linear regressions, we observed R^2 values 0.9869, 0.9919 and 0.9914, for the 34, 99, and re-selected features with p-values of 6.38E-5, 2.41E-5, and 2.75E-5, respectively. Thus, we observed that the model's ability to distinguish between malicious and benign websites decreased over time, specifically, with a rise in FNR.

With RQ11, we investigated the performance of detection models over time. To do so, we first examined the models built from Dataset 1 (see Chapter 7). We found consistency in the number of benign and malicious websites, with approximately 92% of the websites maintaining the same classification over time (based on our RF model built on Dataset 1 with 34 identified features and with 99 features gathered from prior research). Although our model was consistent, it performed poorly. Thus, the model built from Dataset 1 was not able to distinguish between malicious and benign websites. The performance (MCC) of the models was approximately 0, indicating that the classifier was

roughly random and therefore did not provide insight into detection ability over time and was not useful for this study.

In order to create a model that performed well, we re-trained three RF models with different feature sets on the first snapshot of data from Dataset 3 (the Alexa Top 1M domains that were consistent over 12 weeks, with the Google Safe Browsing as the ground truth). That is, we re-trained on the first snapshot of data (gathered February 2nd, 2020) and evaluated the performance on the subsequent weeks. When doing so, we observed an MCC of 0.8960 and an FNR of 0.1906, followed by a performance decrease (a decrease in the MCC) and an increase in the FNR for each subsequent week when using the 34 identified features. The FNR on the last snapshot (taken April 19th, 2020) was 48.46% for the model built from 34 identified features. This result was similar to results for the models built with 99 features and with re-selected features. The decrease in performance was exponential over time without re-training, but by the sixth week of evaluation, we observed FNRs of approximately 30% for the 34 and 41 re-selected features, and of approximately 20% for the 99 features gathered from prior research. These results (and additional results from the other feature sets) indicated that while detection models may be sufficient for the short term, they cannot be guaranteed to work for an extended period of time. This observation aligned with general intuition about the dynamic nature of the internet in an adversarial environment of threat detection. However, the large performance decrease was due to the increasing FNR. The FPR stayed lowed during each iteration (we recorded a max FPR of 0.0032%, 0%, 0.0085%, for the models built from 34, 99, and re-selected features, respectively). This bodes well

for potential incorporation into detectors since triaging false positives is a major problem for security teams [240].

Re-training frequently (in this case, every week) improved performance temporarily, but still resulted in a performance decrease over time with all three feature sets. However, re-training on all prior snapshots resulted in better performing models that remained more robust (their performance decreased more slowly). Additionally, re-training slowed the performance degradation from e^{-t} to t^{-n} in five of the six re-training scenarios based on finding the respective regression with the lowest error. As a result, we postulated that detection models decrease in performance over time, need re-training, and benefit from re-training on various instance of past data.

9.5.2 **RQ12: Do Websites Change Over Time?**

In this next step, we determined whether websites changed over time by examining whether the features that composed those websites changed. To do so, we first examined the pairwise correlations between the respective features to determine any changes. We examined the pairwise correlations between every feature in the respective snapshots (6,160 feature pairs in total) and did so for each feature studied in this chapter. For example, we calculated the correlation between the "Number of HTML tags" feature and the number of `<a>` tags and did this for each of the twelve snapshots. To determine if there were any changes in the pairwise correlations, we looked for outliers as defined by the IQR (Inter Quartile Range) among the sequence of 12 measurements for each pairwise correlation. When doing so on all possible feature pairs, we identified 41 feature pairs (out of 6,160 possible feature pairs) of which each had a single outlier measurement. That is, 41 feature pairs had only a single outlier measurement of the 12

total measurements and the other 6,119 feature pairs had no outlier measurements. Thus, our analysis did not reveal much change in pairwise correlations between the respective snapshots. We next applied four statistical tests to features in the respective snapshots to determine if the feature changed. We did this in a pairwise manner (from the February 2nd snapshot to the February 9th snapshot, from the February 2nd snapshot to the February 16th snapshot, etc.). Results are shown in Figure 9-6 with a significance level of 10%, chosen because we manually inspected the data and observed many p-values just over 0.05 and chose 0.10 (10%) in order to capture these features. We ignored URL features since they do not change.

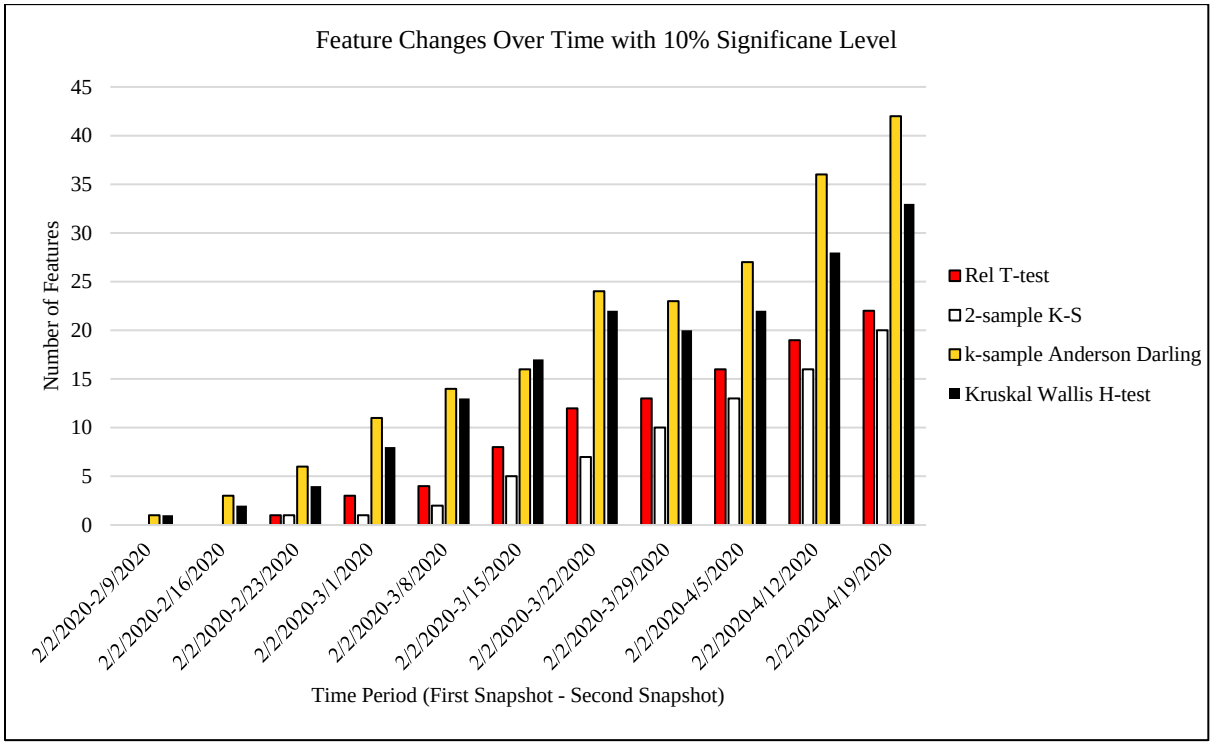


Fig. 9-6. More features changed as the time period lengthened

All four tests showed that the number of changing features increased as the time period lengthened. Although the values were different for all tests, there was an upward trend in the number of features that changed for each test. However, for the two-sample

KS and Kruskal Wallis H test [200]-[201], we observed a greater change for the time period of February 2nd through March 22nd than for the time period of February 2nd through March 29th . All of the data points compared the features collected on February 2nd to the end date separated by a "-."

In addition to measuring the number of features that changed over the respective time periods, we also examined the feature importance associated with the respective features that changed. Figure 9-7 shows the total importance of the features that changed within the three feature sets between February 2nd and April 19th.

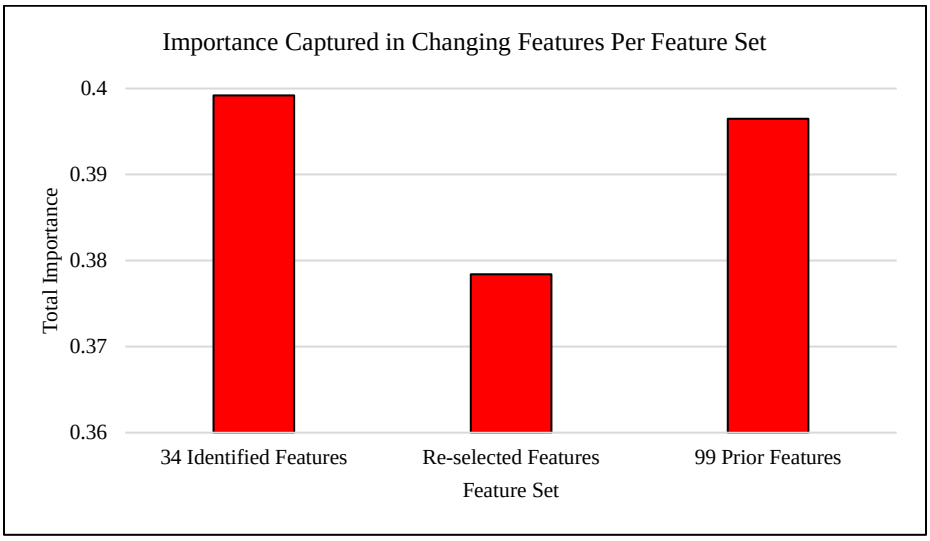


Fig. 9-7. The features that change represented more than 1/3 of total feature importance

Of the features that changed over the time period measured, approximately 38%–40% of the total feature importance was contained in these features across the three feature sets. Thus, the features that changed were influential in determining whether or not a website was malicious in that they captured non-trivial amount of feature importance. Additionally, approximately 24%-37% of the features in the respective datasets were URL features (which do not change over time). Figure 9-8 shows the cumulative importance of URL features.

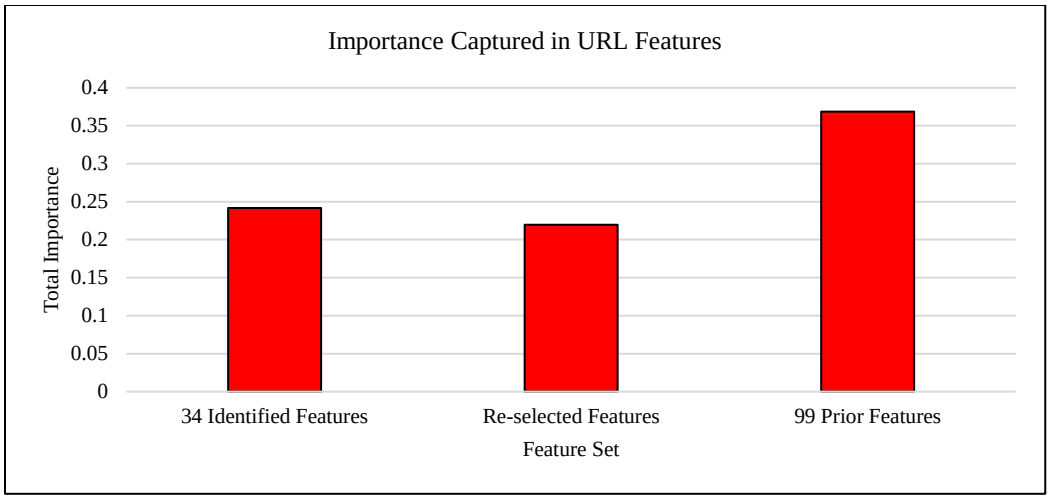


Fig. 9-8. More than 20% of total feature importance was derived from URL features

After identifying the total importance from the features that changed over time, we then examined how much total importance changed over each week. This was calculated by examining the features that changed during each time period and summing their importance for each interval. Results are shown in Figure 9-9, based on the related t-test.

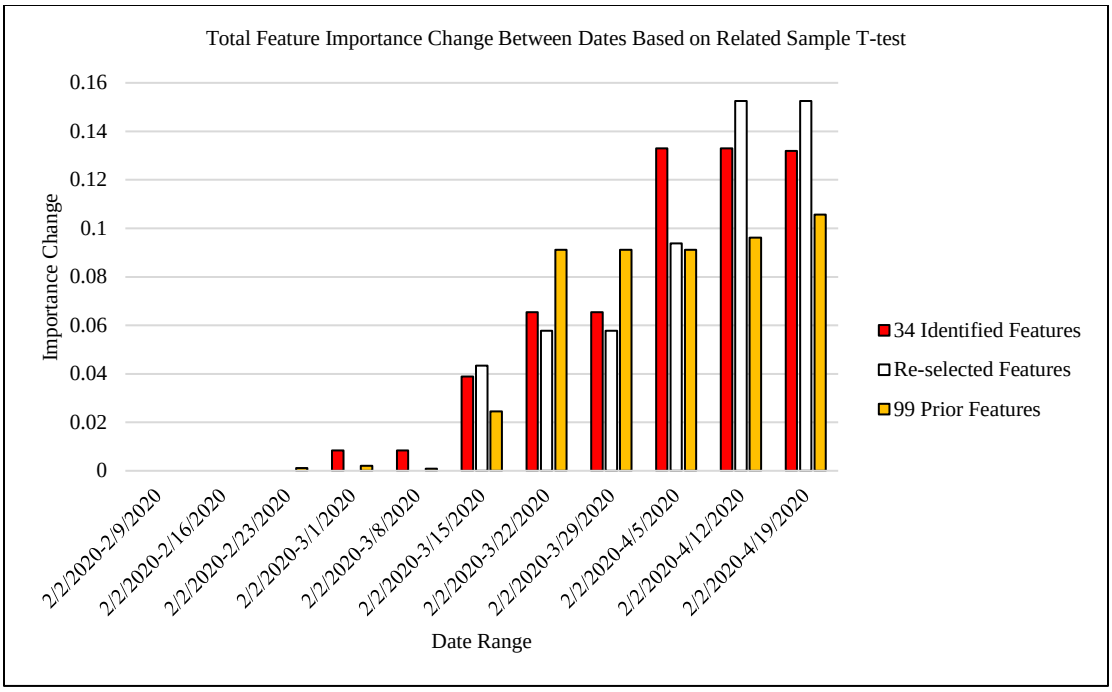


Fig. 9-9. Feature importance changed more as the time gap became larger when using the related sample t-test

Results from this test showed less than a 0.02 change in feature importance from February 2nd, 2020 to March 8th, 2020 over the three feature sets within the first few weeks. We then examined the feature importance that changed when we considered as "changed" those features that changed on at least one of the four tests. This is shown in Figure 9-10.

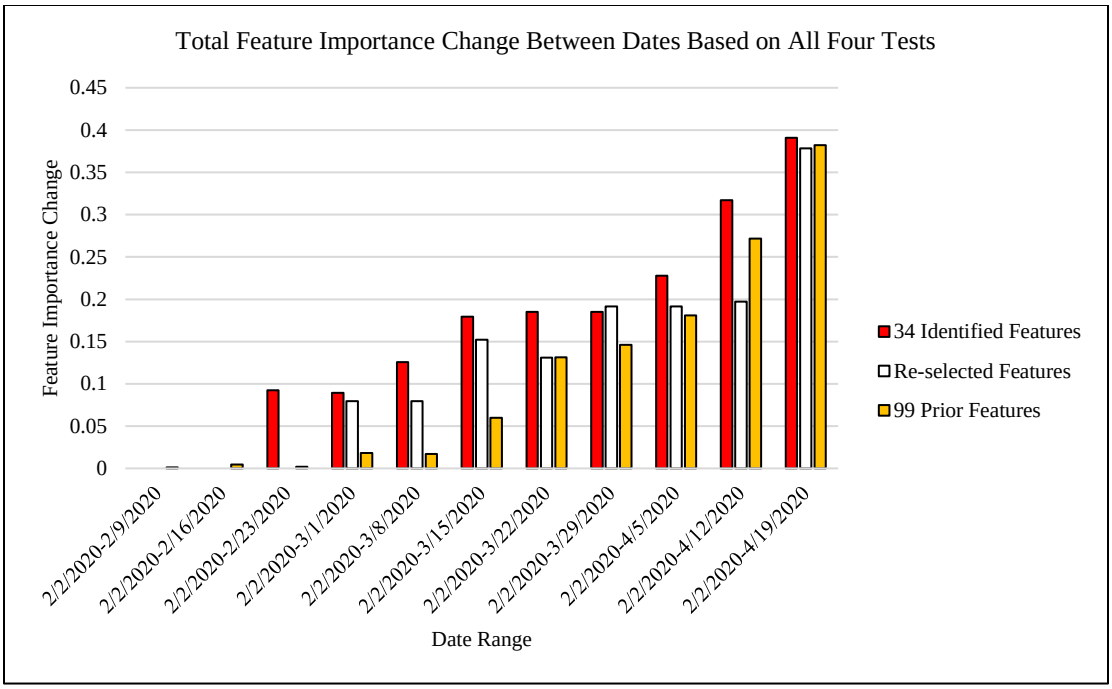


Fig. 9-10. When using several tests, feature importance changed more as the time gap increased

The results shown in Figure 9-10 are similar to the results shown in Figure 9-9 in that the first few weeks showed little importance change followed by larger importance change in the last weeks. Thus, we observed that the features that compose websites do change and that those features accounted for roughly 40% of the total feature importance in the respective feature sets, confirming that websites do change over time.

With RQ12, we investigated whether websites changed over time. We focused the analysis on the features that compose a website and on those used for discriminating between malicious and benign websites. All four statistical tests demonstrated that the number of features that change over time increased. Additionally, we found that the features that change over time accounted for a non-trivial amount of feature importance in our detection model. Thus, we postulated that websites, as defined by their features that can be used for malicious website detection, change over time.

9.5.3 RQ13: If Websites Change Over Time, How Much do They Change Over Time?

To answer RQ13, we used techniques and tests similar to those used in RQ12, but we measured the results as a function of the number of weeks that had passed. In other words, instead of comparing each week to the first week, as we did when answering RQ12, we compared each week to each of the other weeks. For example, we performed analysis on the data and changes between the weeks of March 2nd and March 9th, the weeks of March 2nd and March 16th, the weeks of March 2nd and March 23rd, etc. As a result, we obtained a series of measurements as a function of the number of weeks. We obtained 11 measures where the week difference was 1 week, 10 measurements where the week difference was 2 weeks, 9 measurements where the week difference was 3 weeks, etc.

First, we examined the average number of features that changed over time as a function of the week difference. Results are shown in Figure 9-11.

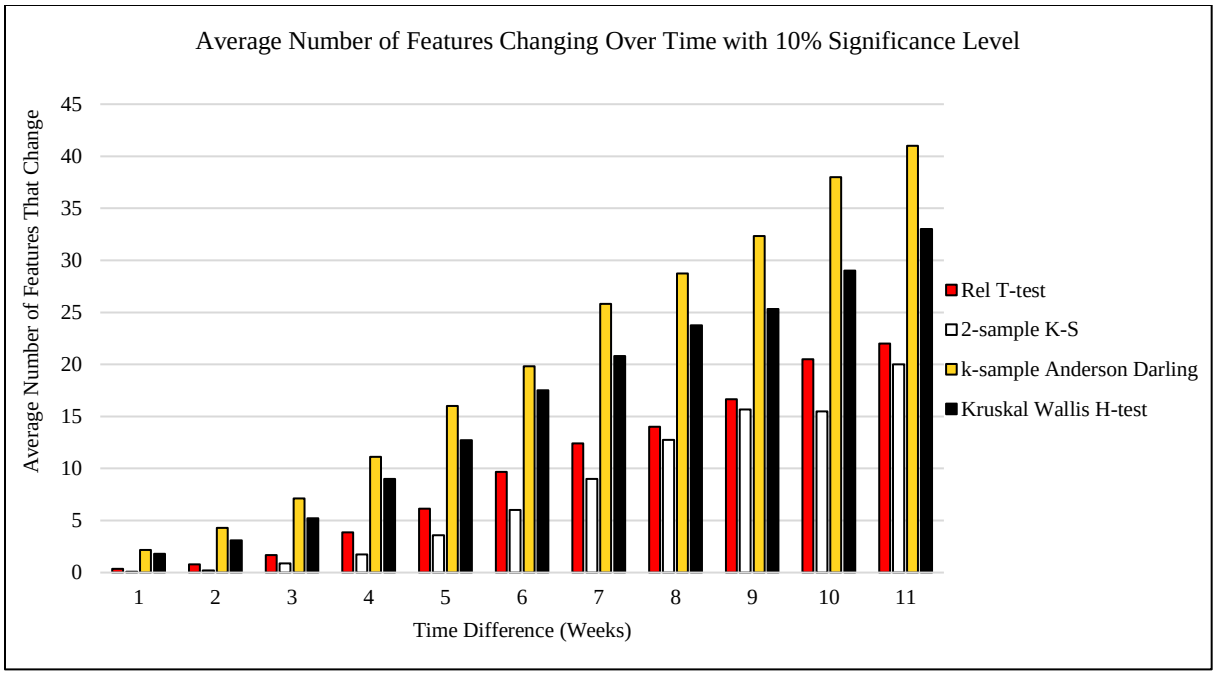


Fig. 9-11. The average number of features that changed over time increased with the lengthening of the time period

As with the findings from RQ12, Figure 9-11 showed the constant upward trend we observed in the number of features that changed over time. Box plots of the respective feature changes based on the respective tests are shown in Figures 9-12, 9-13, 9-14, and 9-15.

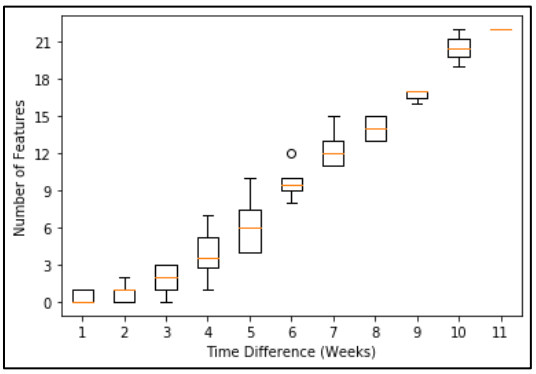


Fig. 9-12. Box plot for the number of features that changed over time, per related sample t-test

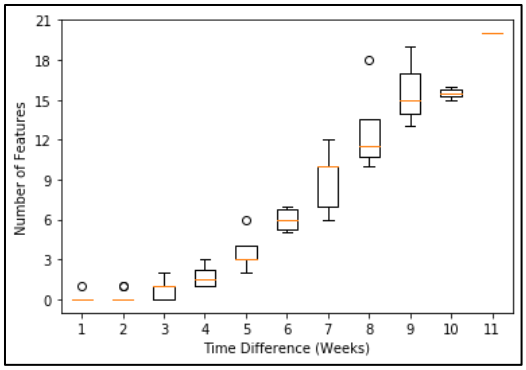


Fig. 9-13. Box plot for the number of features that changed over time, per two-sample KS

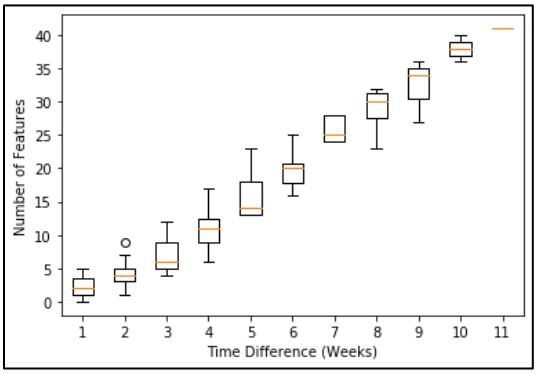


Fig. 9-14. Box plot for the number of features that changed over time, per k-sample Anderson-Darling

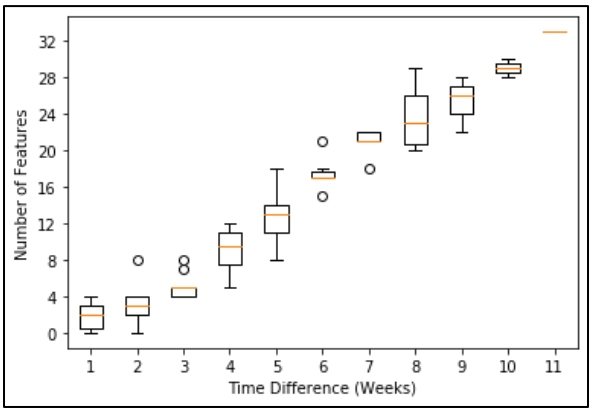


Fig. 9-15. Box plot for the number of features that changed over time, per the Kruskal Wallis H test

We observed overlap in the box plots for the number of features that changed when the time difference was one week and two weeks. Although we made this observation, the fact that a similar number of features changed when the time difference

was one or two weeks did not appear to have improved detection results since there was no overlap in the box plots for the performance as a function of the time difference (with the exception of a few outlier measurements). Results are shown in Figures 9-16, 9-17, and 9-18 below for the 34, re-selected, and 99 features, respectively.

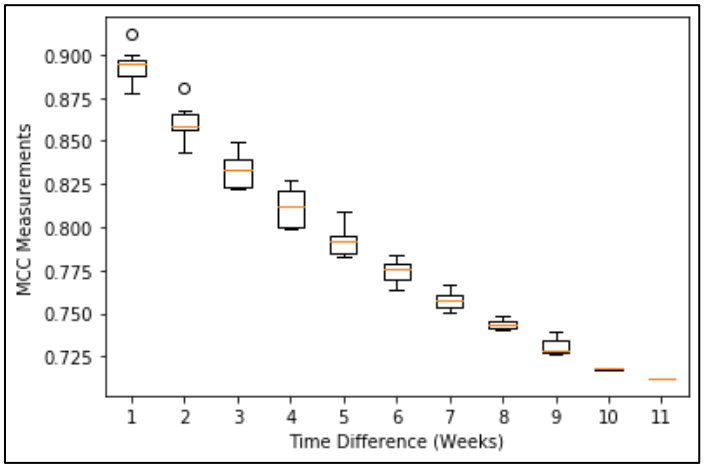


Fig. 9-16. Capturing several measurements as a function of time further demonstrated performance decrease when using 34 features for malicious website detection

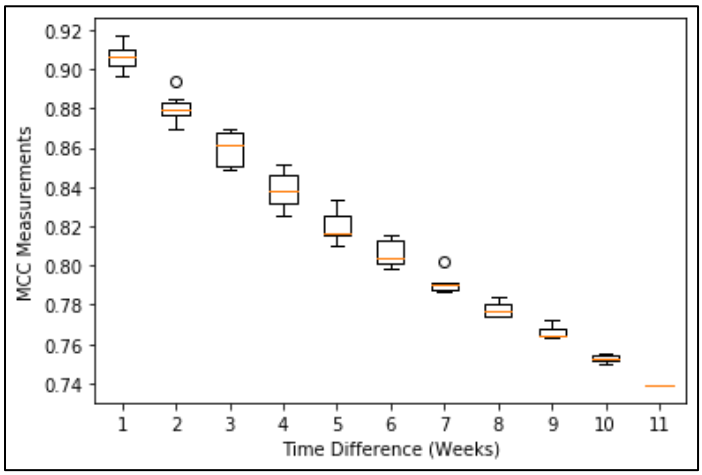


Fig. 9-17. Capturing several measurements as a function of time further demonstrated performance decrease when using re-selected features for malicious website detection

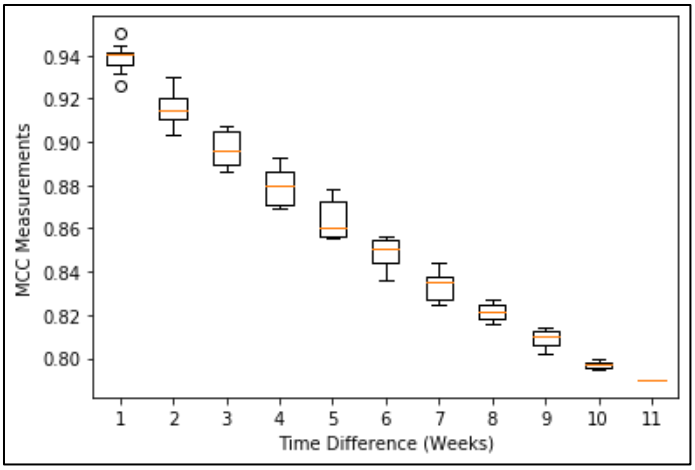


Fig. 9-18. Capturing several measurements as a function of time further demonstrated performance decrease when using 99 prior features for malicious website detection

We investigated the number of features that changed by analyzing all possible pairs. Findings further supported the conclusion that features changed over time and that the change followed an upward trend while performance followed a downward trend.

With RQ13, we verified our initial observations from RQ12 by performing an additional analysis and calculating the results when comparing the features over every possible combination of snapshots to gauge how much websites changed over time and if the change was consistent. We knew that websites are updated over time, but to our knowledge, this is the first study that attempted to evaluate how websites change with regard to malicious website detection. We observed that the longer the time period between model training and model use, the worse the performance of the original model and the more features changed. This finding supported our observation from RQ11 regarding the need to frequently re-train. As features begin to change, models will become stale and experience changing performance metrics. We observed a decrease. Thus, we demonstrated that websites do change over time and that the changes become

larger as the timeframe increases, a finding that highlights the need to re-train detection models.

9.6 Conclusion

This chapter included a temporal evaluation of feature-based malicious website detection. In this chapter, we detailed our investigation into whether detection models remained effective over a period of time and the different strategies we used for re-training. Additionally, we provided insight into whether and how websites changed over time and demonstrated that as websites change (in the form of their features), the performance of detection models consistently decreased without re-training.

Chapter 10: Limitations

This dissertation included several studies on feature-based malicious website detection. However, it was not without limitations. In this chapter we discuss the limitations present throughout this research.

10.1 Dataset Selection

The first limitation originated from dataset selection. In related security research, authors used different techniques to create datasets of benign and malicious websites. Some authors implemented web crawlers or used randomly selected URLs or similar methods to identify and collect websites to supplement or define their datasets [35]-[37], [40], [42]-[44], [46], [49]-[50], [81], [91], [95]. Others used established or well-known datasets as part of their datasets [24], [47]-[48], [86], [89], [116]-[117]. Both approaches to defining and curating datasets of benign and malicious websites include inherent subjectivity. We based our datasets (Dataset 1 used in Chapters 4-7, Dataset 2 used in Chapter 8, and Dataset 3 used in Chapter 9) from external sources – the Alexa top 1M, Cisco Talos Intelligence Group, Cymon.io, and Google Safe Browsing in an attempt to be objective and lessen our influence in our studies.

In our studies in Chapters 4-7, we assumed that popularity and high Alexa rank were benign traits, though this may not always be true. To investigate our assumption, we verified with threat intelligence feeds from Cymon.io [193], a tool that accumulates threat intelligence. In 2018, we observed that approximately 5% of the websites in our benign list appeared in the Cymon.io database. While the appearance in, or absence from, the Cymon.io database does not confirm the benign or malicious nature of the website, given that only 5% of our benign websites were present in Cymon.io, we demonstrated that the

assumption of popularity as a trait of benign websites was reasonable. This observation suggests that at most 5% of our benign data was mislabeled. Also, our dataset in Chapters 4-7 represented a specific point in time. Internet security and the web are ever-changing environments, providing no guarantee that our findings will remain true should this experiment be repeated on a different dataset. This limitation is difficult to avoid in dynamic environments like the internet. We did however address this limitation with our study in Chapters 8 and 9.

For purposes of our study, a website was considered "malicious" if it was associated with any attacks including phishing, drive-by downloads, or C2 infrastructure. "Malicious" does not have a precise, standardized definition in a cybersecurity context, so definitions may vary. Therefore, we run the risk of disagreeing with other researchers who may define "malicious" differently.

10.2 Feature Challenges

There are some limitations present in the features themselves. Webpage content provides a rich environment for feature collection, a fact that we took full advantage of in conducting our research. However, the extent to which HTML and JavaScript can be studied is vast, and some methods from previous research present challenges when attempting to combine many different analysis techniques. For example, HTML can contain many URLs. Although we analyzed properties of these URLs in our collection, URL analysis itself is vast, encompassing several detection means that were not compatible with our approach. The JavaScript on the webpage posed the same challenge. Our approach was static and therefore did not include the several dynamic approaches to JavaScript that exist. Thus, feature collection and analysis for webpage content is

challenging as a result of the many different features that can be collected and as a result of the many different analysis techniques.

The gathering of JavaScript features posed another limitation. We gathered our JavaScript features statically, which has been done in prior research, but because malicious JavaScript is often obfuscated, it presents a challenge to analysis. Potential mitigations include adding a de-obfuscator or instrumenting the collection environment to record the specific JavaScript methods executed. This requires additional overhead and potentially runs the risk of executing malicious script while attempting to perform feature collection.

Additionally, our set of URLs consisted of English URLs, a choice that greatly influenced the lexical features we extracted in our research. Should our dataset have contained URLs with non-English characters, we would have needed to modify our feature set and collection mechanism to account for this. URL features are very flexible since URLs consist simply of strings of characters. As a result, they can be analyzed in many ways. Given this flexibility, there was a risk that our approach – examining n-grams on the URL – may not have been the optimal approach for analyzing URLs. Given the existence of many different analysis techniques, it is challenging to identify the single best analysis technique.

The selection of HTTP header features also limited us. HTTP header analysis requires substantial data cleaning and validation due to the prevalence of custom headers, misspellings, and so on. For our exploration into HTTP header features and their applicability to detect malicious websites, we focused on collecting and cleaning headers received in a `response` to a `GET request`. While this provided a rich set of features, we

did not collect session-based features or those features arising from HTTP `requests` and `responses` over a period of time. In addition, we used HTTP features in isolation, rather than in combination with other website features.

10.3 Comparison with Other Works

Benchmarking our work to related research was a challenge. Prior researchers used different datasets, features, and performance metrics, collected their data at different times, or focused on different aspects, such as the speed of website classification or other metrics. This points to a broader problem in the field of cybersecurity – a lack of repeatability – that hinders validation and comparison.

10.4 Additional Limitations

The last limitations came from our last two chapters where we explored additional datasets and performed a temporal evaluation of malicious website detection. The main limitation but also key finding in Chapter 8 was that the ability to apply feature-based detection to malicious websites was dependent upon the datasets themselves. After we demonstrated that we could not apply a model built with one dataset to another, we then observed differences with correlations and association between the features studied on the respective datasets. If we had used two similar datasets, our results most likely would have demonstrated better detection. Although this was a challenge, this observation also is key to assessing the real-world application of this method.

For Chapter 9, we chose a dataset source (the Alexa 1M) that is leveraged in various studies, and ensured consistency by only studying websites that appear in the Alexa Top 1M in each of the 12 weeks. Although our dataset has objective rationale, there was some subjectivity in choosing a dataset. Additionally, we scoped our study to a

period of 12 weeks or approximately 90 days. From prior research we observed that detection performance decreases after 1-3 months and this is the basis for the 12-week period of this study. Studies on website change have also spanned approximately three months. Although the 12-week duration was based on prior research, it too was somewhat subjective.

Another limitation with Chapter 9 resulted from the notion of measuring website changes over time. Because little work has been completed in the field with respect to malicious website detection change over time, we found no agreed-upon method for analysis. Furthermore, few statistical tests are designed for measuring website change. We chose four tests that appeared to be the most appropriate and their results were similar, though the lack of a universally agreed-upon method and test for measuring website change posed a challenge.

Lastly, the dynamic nature of the internet created a limitation for our research in Chapter 9. Some websites change quickly, while others change more gradually. We use a week-to-week analysis which is based on an observation from prior research that websites are likely to change within a week, however acknowledge the subjectivity of this frequency. We began each weekly collection on a specific date, but since the collection of data for hundreds of thousands of websites cannot happen instantly, actual timestamps of collection were not identical.

Chapter 11: Conclusions

11.1 Dissertation Summary

Researchers have extensively used website features to detect malicious websites. With this research, we performed a comprehensive evaluation of feature-based malicious website detection. First, we reviewed prior research that established features that are relevant for malicious website detection, leveraged detection methods (heuristics, machine learning, etc.), presented potential validation methods, provided practical implementations, discussed relevant performance metrics, and evaluated website change over time. In Chapter 3, we presented our methodology and the 13 research questions that drove it. In Chapters 4–6, we presented independent studies on malicious website detection using three separate classes of features, validating prior research as well as presenting new findings. In Chapter 7 we leveraged the findings and features from Chapters 4–6, going on to evaluate the discovery of features through feature selection versus using those from gathered prior research. In Chapter 8 we reported our application of detection models built on one dataset to another dataset, while in Chapter 9 we detailed our temporal study on feature-based malicious website detection.

We established that feature-based malicious website detection remains relevant for detection of several types of threats and that re-evaluation of the assumptions from prior research (including the features used for detection) yields benefits. Our research showed improvement when using discovered features versus features gathered from prior research. This improvement was demonstrated with models built from various machine learning algorithms over various scenarios. Furthermore, we demonstrated that feature selection (versus selecting features in advance) decreased the number of features needed

for malicious website detection. The study in each chapter that demonstrated the benefit of building detection models with new features was performed with a dataset consisting of several threats. Furthermore, the features that were used for detection are available in a web browsing environment. Thus, we recommend the addition and exploration of new features in future research.

In our last two chapters, we evaluated feature-based malicious website detection on two additional datasets. By doing so, we showed that detection models were reliant on the dataset on which they trained, however, the features that we identified could be applied to new datasets with minor adjustments. Our study of the temporal aspects of malicious website detection provided evidence that malicious website detection models degrade over time. Re-training can improve model performance and can slow performance degradation. From the results in Chapter 8 and Chapter 9, we postulate that adjusting models with new features (as done in Chapter 8) and retraining as new data becomes available (Chapter 9) will improve malicious website detection. Lastly, we presented a method of quantifying how websites (as defined by their features) change over time and quantified the change we observed.

11.2 Future Work

There are two potential areas of future work that could follow this dissertation.

One possibility is the specification, creation, and maintenance of a central dataset for malicious website detection experiments. Several datasets are used in prior research that differ in size, types of threat, ratio of malicious to benign websites, and date of collection. These differences make comparison of prior research and techniques challenging because each researcher typically uses a dataset specific to their study.

Specifying, creating, and maintaining a training and evaluation dataset including both malicious and benign websites would be beneficial to the research community.

Another possibility of future work involves the creation and evaluation of the research in this dissertation into a potential security tool. That is, follow on work could potentially involve building a component inside of a web browser or other tool that fetches websites. There are several areas to be addressed in this work. First, we can investigate different sources of training data and evaluate their effectiveness in an operational environment. The training data could be open source intelligence, data from a security operations centers, data from the users of the tool, or potentially a combination thereof. Second, we could evaluate the utility of using this tool as a blocking mechanism (preventing users or services from accessing a website) or as an aid to a user or a service making browsing decisions. By evaluating this solution as a blocking mechanism, we would gain insight into the usability of such a solution – particularly, is the false positive rate low enough to prevent disruption. By evaluating this capability as a supplement to a user, we would gain insight into if and how this mechanism benefits from user input. Additionally, we could gauge if this capability enables a user or service to make beneficial risk-based decisions on whether or not to visit a website.

Lastly, the completion of this dissertation involved the creation of various scripts and software components. We are currently working on the release and sharing of code used in this dissertation.

Appendices

Appendix A: URL Features

1. Presence of an IP address in the URL
2. Presence of a port number in the URL
3. Presence of a well-known ports in the URL
4. The length of URL
5. Counts of each character
6. Total count of digits
7. Total count of letters
8. Total count of special characters
9. Counts of n-grams from files extensions in the URL
10. Total count of file extension n-grams in the URL
11. Counts of n-grams of TLDs in the URL
12. Total count of file extension n-grams in the URL
13. Count of respective words from [210]
14. Count of the number of words in the URL with a given length

Appendix B: JavaScript Methods

We extracted the counts of the following methods within the webpage.

DOM Methods:

adoptNode
captureEvents
caretPositionFromPoint
caretRangeFromPoint
clear
close
createAttribute
createAttributeNS
createCDATASection
createComment
createDocumentFragment
createElement
createElementNS
createEntityReference
createEvent
createNodeIterator
createProcessingInstruction
createRange
createTextNode
createTouch
createTouchList
createTreeWalker
elementFromPoint
elementsFromPoint
enableStyleSheetsForSet
execCommand
exitPointerLock
getAnimations
getElementById
getElementByName
getElementsByClassName
getElementsByTagName
getElementsByTagNameNS
getSelection
hasFocus
importNode
normalizeDocument
open
queryCommandEnabled
queryCommandIndeterm
queryCommandValue
querySelector
querySelectorAll
releaseCapture
releaseEvent
routeEvent
write
writeln

Java Script:

add
all
allTrue
anchor
apply
big
bind
blink
bold
call
catch
charAt
charCodeAt
clear
codePointAt
compile
concat
construct
copyWithin
defineProperty
delete
deleteProperty
endWith
entries
every
exec
fill
filter
finally
find
findIndex
fixed
flat
flatMap
fontcolor
fontsize
forEach
formatToParts
from
get
getDate
getDay
getFloat32
getFloat64
getFullYear
getHours
getInt16
getInt32
getInt8
getMilliseconds
getMinutes
getMonth
getOwnPropertyDescriptor
getPrototypeOf
getSeconds
getTime
getTimezoneOffset
getUTCDate
getUTCDay
getUTCFullYear
getUTCHours
getUTCMilliseconds
getUTCMinutes
getUTCMonth
getUTCSeconds
getUint16
getUint32
getUint8
getYear
grow
has
hasOwnProperty
includes
indexOf
isExtensible
isNan
isPrototypeOf
italics
join
keys
lastIndexOf
link
localeCompare
log
map
match
min
next
normalize
of
ownKeys
padEnd
padStart
pop
preventExtensions
propertyIsEnumerable
push
reduce
reduceRight
repeat
replace
resolvedOptions
return
reverse
search
select
set
setDate
setFloat32
setFloat64
setFullYear
setHours
setInt16
setInt32
setInt8
setMilliseconds
setMinutes
setMonth
setPrototypeOf
setSeconds

setTime
setUTCDate
setUTCFullYear
setUTCHours
setUTCMilliseconds
setUTCMinutes
setUTCMonth
setUTCSeconds
setUint16
setUint32
setUint8
setYear
shift
slice
small
some
sort
splice
split
strike
subarray
substr
substring
supportedLocalesOf
test
throw
toDateString
toExponential
toFixed
toISOString
toJSON
toLocaleString
toLocaleTimeString
toLocaleUpperCase
toLowerCase
toPrecision
toSource
toString
toUpperCase
trim
trimEnd
trimStart
unshift
valueOf
values

Array.from
Array.isArray
Array.of
ArrayBuffer.isView
ArrayBuffer.transfer
Atomics.add
Atomics.and
Atomics.compareExchange
Atomics.exchange
Atomics.isLockFree
Atomics.load
Atomics.or
Atomics.store
Atomics.sub
Atomics.wait
Atomics.wake
Atomics.xor
Date.UTC
Date.now
Date.parse
Intl.getCanonicalLocales
JSON.parse
JSON.stringify
Math.abs
Math.acos
Math.acosh
Math.asin
Math.asinh
Math.atan
Math.atan2
Math.atanh
Math.cbrt
Math.ceil
Math.clz32
Math.cos
Math.cosh
Math.exp
Math.expm1
Math.floor
Math.fround
Math.hypot
Math.imul
Math.log
Math.log10
Math.log1p
Math.log2
Math.max
Math.min
Math.pow
Math.random
Math.round
Math.sign
Math.sin
Math.sinh
Math.sqrt
Math.tan
Math.tanh
Math.trunc
Number.isFinite
Number.isInteger
Number.isNaN
Number.isSafeInteger
Number.parseFloat
Number.parseInt
Object.assign
Object.create
Object.defineProperties
Object.defineProperty
Object.entries
Object.freeze
Object.getOwnPropertyDescriptor
Object.getOwnPropertyDescriptors
Object.getOwnPropertyNames
Object.getOwnPropertySymbols
Object.getPrototypeOf
Object.is
Object.isExtensible
Object.isFrozen
Object.isSealed
Object.keys
Object.preventExtensions
Object.seal
Object.setPrototypeOf
Promise.all
Promise.race
Promise.reject
Promise.resolve
Proxy.revocable
Reflect.apply
Reflect.construct
Reflect.defineProperty
Reflect.deleteProperty
Reflect.get
Reflect.getOwnPropertyDescriptor
Reflect.getPrototypeOf
Reflect.has
Reflect.isExtensible
Reflect.ownKeys
Reflect.preventExtensions
Reflect.set
Reflect.setPrototypeOf
String.fromCharCode
String.fromCodePoint
String.raw
Symbol.for
Symbol.keyFor
WebAssembly.compile
WebAssembly.compileStreaming
WebAssembly.customerSections
WebAssembly.exports
WebAssembly.imports
WebAssembly.instantiate
WebAssembly.instantiateStreaming
WebAssembly.validate
decodeURI
decodeURIComponent
encodeURI
encodeURIComponent
escape
eval
isFinite
isNaN
parseFloat
parseInt
undefined
unescape
uneval

Methods on the Window Object:

addEventListener
alert
atob
back
blur
btoa
cancelAnimationFrame
cancelIdleCallback
captureEvents
clearImmediate
clearInterval
clearTimeout
close
confirm
createImageBitmap
disableExternalCapture
dispatchEvent
dump
enableExternalCapture
fetch
find

focus
forward
getAttention
getAttentionWithCycleCount
getComputedStyle
getDefaultComputedSyle
getSelection
home
matchMedia
maximize
minimize
moveBy
moveTo
open
openDialog
postMessage
print
prompt
releaseEvents
removeEventListener
requestAnimationFrame
requestIdleCallback
resizeBy
resizeTo
restore
routeEvent
scroll
scrollBy
scrollByLines
scrollByPages
scrollTo
setCursor
setImmediate
setInterval
setResizable
setTimeout
sizeToContent
stop
updateCommands

Initial Tag Counts Counted

a	command	hr	noembed	spacer
abbr	content	html	noframes	span
acronym	data	i	noscript	strike
address	datalist	iframe	object	strong
applet	dd	image	ol	style
area	del	img	optgroup	sub
article	details	input	option	summary
aside	dfn	ins	output	sup
audio	dialog	isindex	p	table
b	dir	kbd	param	tbody
base	div	keygen	picture	td
basefont	dl	label	plaintext	template
bdi	dt	legend	pre	textarea
bdo	element	li	progress	tfoot
bgsound	em	link	q	th
big	embed	listing	rp	thead
blink	fieldset	main	rt	time
blockquote	figcaption	map	rtc	title
body	figure	mark	ruby	tr
br	font	marquee	s	track
button	footer	menu	samp	tt
canvas	form	menuitem	script	u
caption	frame	meta	section	ul
center	frameset	meter	select	var
cite	h1	multicol	shadow	video
code	head	nav	slot	wbr
col	header	nextid	small	xmp
colgroup	hgroup	nobr	source	

Element Attribute Features

For the following attributes, we collected deeper information that fell into three categories.

1. Certain attributes specify resources via URLs. For these attributes, we extracted whether the reference pointed to an OoD resource, an in-domain resource, a relative link within the page, and the protocol specified by the resource. We also captured the protocol specifying the location to the resource. Additionally, these resources are typically of a certain file type and we collected which file type as well.
2. Certain elements can be of a "small size" or be a "small element." We defined a small element as one that had a length or width of fewer than 2 pixels.

The matrix below shows which additional attributes we extracted from the respective HTML elements.

	accept	accept-charset	async	autoplay	charset	crossorigin	datasrc	decoding	defer	download	enctype	hidden	hreflang	http-equiv	lang	mayscript	referrerpolicy	rel	sandbox	srclang	small element	type	url info
a					X					X		X	X		X		X	X				X	href
applet							X					X			X			X			X	X	src,codebase,archive
area					X					X		X	X		X		X	X				X	href
audio												X			X			X				X	src,codebase,archive
base												X			X			X				X	href
bgsound												X			X			X					src
canvas												X			X			X			X	X	
embed												X			X			X			X	X	src
form	X	X									X	X			X			X				X	action
frame												X			X			X				X	src
iframe												X						X	X		X	X	src, longdesc
img						X		X				X			X		X	X			X	X	src, srcset
link					X	X						X	X		X			X				X	href
main					X							X			X			X				X	
meta					X							X		X	X			X				X	
object					X							X			X			X				X	codebase, data,archive
script			X		X	X			X						X			X				X	src
source												X						X				X	src, srcset
track												X			X			X		X		X	src
video				X		X						X			X			X			X	X	src

Appendix D: Full Tables and Charts

Table D-1:
The Count of "-" Characters Had High Correlation with the Target Variable

Correlation Values between Target Variable and 288 Features on Dataset 2	
Features	***Dataset 2***
Count of '-' character	0.3660
marco contenttext	0.0094
content-language text/html	0.0081
URL Length	0.0188
<link href="https*">	0.0252
<link rel="canonical">	0.0240
<link href> OoD	0.0173
<script src> relative	0.0150
<script src> absolute	0.0150
Count of 'w' character	0.0022
<link rel=https://api.w.org/*>	0.0248
<script src="https*">	0.0182
<link rel="stylesheet">	0.0172
<link rel="wlwmanifest">	0.0243
<link type="application/wlwmanifest+xml">	0.0243
<link type="application/rsd+xml">	0.0245
<link rel="EditURI">	0.0245
<meta http-equiv="X-UA-Compatible">	0.0130
<script type="text/javascript">	0.0174
Total URL Extensions	0.0006
Count of 'u' character	0.0199
<script src> OoD	0.0153
URL TLD "co" Count	0.0257
<link rel="dns-prefetch">	0.0219
<link rel="shortlink">	0.0243
content-encoding gzip	0.0027
<link type="text/css">	0.0148
URL extension is ".c"	0.0277
<link href="*.xml">	0.0164
URL extension ".com"	0.0205
URL TLD "com"	0.0205
<script async=true>	0.0100
meta_charset_UTF-8	0.0154
<link rel="icon">	0.0120
Count of '.' Character	0.0159
<script src="*.js">	0.0131
<link rel="pingback">	0.0188
getElementsByTagName()	0.0085
<link href> absolute	0.0085
<link href> relative	0.0085
Count of <link> tag	0.0085
<link href=*.png>	0.0100
Count of <script> tag	0.0070
<meta http-equiv="content-type">	0.0234
Count of 'x' character	0.0062
expect-ctmax-age	0.0137
expect-ctreport-uri	0.0138
<link href="*0">	0.0150
Count of 'z' character	0.0074
vary accept-encoding	0.0069
vary accept	0.0068
via 1.1	0.0135
URL endswith ".com"	0.0154
Count of 'f' character	0.0057

Correlation Values between Target Variable and 288 Features on Dataset 2	
Features	***Dataset 2***
<script type="application/ld+json">	0.0177
<link href="*.css">	0.0055
Count of 'a' character	0.0262
createElement()	0.0040
x-xss-enabled	0.0142
x-cintent-type-options nosniff	0.0137
<link rel="publisher">	0.0098
Count of 4-character words	0.0314
<link rel="shortcut icon">	0.0007
Count of tag	0.0031
<link rel="manifest">	0.0095
Count of <iframe> tag	0.0018
<link rel="apple-touch-icon-precomposed">	0.0118
<link rel="apple-touch-icon">	0.0053
indexOf()	0.0070
<link href="*.php">	0.0076
<img srcset="https*">	0.0091
Count of 'o' character	0.0192
Count of 'e' character	0.0073
Count of <a> tag	0.000012
<a href> relative	0.000036
<link href="*.json">	0.0062
content-length	0.0040
<a href> absolute	0.0001
Count of 'c' character	0.0182
Count of <img> tag	0.0005
Count of 'p' character	0.0055
Total href attributes	0.0001
Total HTML Tags	0.0002
Count of 'm' character	0.0110
<a href="https*">	0.0041
Count of 'r' character	0.0146
<script_src="*.0">	0.0107
<img srcset> absolute	0.0084
<img srcset> relative	0.0084
<script defer=true>	0.0110
setTime()	0.0058
Count of <nav> tag	0.0049
Count of <style> tag	0.0056
<link type="application/rss+xml">	0.0115
isNaN()	0.0008
<link rel="alternate">	0.0047
<iframe src="https*">	0.0024
<meta charset=utf-8>	0.0154
<link_type="image/png">	0.0046
getTime()	0.0075
cache-control max-age	0.0108
apply()	0.0095
getElementById()	0.0058
match()	0.0073
strict-transport-security max-age	0.0060
server nginx	0.0219
Count of 'v' character	0.0066
<a href> OoD	0.0019
Count of <footer> tag	0.0045
replace()	0.0058
<iframe src> absolute	0.0007
vary age	0.0082
addEventListener()	0.0093
<img srcset> OoD	0.0073
vary user-agent	0.0083
<iframe src> relative	0.0003
Count of <header> tag	0.0043

Correlation Values between Target Variable and 288 Features on Dataset 2	
Features	***Dataset 2***
JSON.parse()	0.0050
vary cookie	0.0101
Count of tag	0.0025
bind()	0.0006
Count of 'y' character	0.0103
Count of tag	2.1119
<a href-="*.aspx>	0.0064
JSON.stringify()	0.0048
trim()	0.0030
Count of <main> tag	0.0027
transfer-encoding chunked	0.0102
log()	0.0055
<a href=script:javascript>	0.0032
connection keep-alive	0.0109
Count of tag	0.0005
toLowerCase()	0.0050
<link href="http*">	0.0012
hasOwnProperty()	0.0053
Count of 5-character words	0.0160
<script src="*.com">	0.0050
Count of <i> tag	0.0001
Total TLDs in URL	0.0303
Count of 'h' character	0.0080
<iframe_src="*.com">	0.0018
encodeURIComponent()	0.0042
setTimeout()	0.0034
Count of 'i' character	0.0096
Count of <section> tag	0.0033
concat()	0.0086
decodeURIComponent()	0.0006
Count of <meta> tag	0.0048
Math.random()	0.0015
Count of <time> tag	0.0019
Count of 'd' character	0.0015
<a href="*.htm">	0.0006
<link href="*.ico">	0.0037
Count of <button> tag	0.0006
<link rel="next">	0.0037
escape()	0.0196
<img src="https*">	0.0042
Count of 't' character	0.0037
cache-control no-store	0.0102
test()	0.0048
Count of tag	0.0018
join()	0.0086
Count of tag	0.0004
Count of <article> tag	0.0036
url_extension_.i	0.0069
cache-control must-revalidate	0.0084
Count of 'n' character	0.0081
<link_href="*.com">	0.0021
cache-control private	0.0089
<a href="mailto*">	0.0027
Count of <select> tag	0.0016
Count of <form> tag	9.0816
Math.floor()	0.0001
split()	0.0054
Count of <hr> tag	2.4394
url_extension_.net	0.0011
url_tld_NET	0.0011
<img src> relative	0.0040
<img src> absolute	0.0040
<a href="http*">	0.0062

Correlation Values between Target Variable and 288 Features on Dataset 2	
Features	***Dataset 2***
<img src=”0*”>	0.0036
Count of ‘s’ character	0.0001
pop()	0.0079
Count of 7-character words	0.0019
Count of <noscript> tag	0.0023
URL TLD “ne”	0.0009
<iframe src> OoD	0.0017
substring()	0.0025
<link type=”image/x-icon”>	0.0014
<form enctype=”application/ x-www-form-urlencoded”>	0.0013
Count of <small> tag	0.0013
Count of <ins> tag	0.0008
<img_src=”*.jpg”>	0.0057
Total <img src>	0.0050
substr()	0.0016
server apache	0.0043
exec()	0.0023
parseInt()	0.0007
URL extension “.net”	0.0011
Count of <dl> tag	0.0029
push()	0.0008
open()	0.0021
<a rel=”nofollow”>	0.0036
<link rel=”mask-icon”>	0.0048
Count of <figure> tag	0.0003
<form action> relative	0.0013
<form action=”https*”>	0.0010
find()	0.0021
Count of <option> tag	0.0013
<form action>_absolute	0.0015
shift()	0.0078
<base href> OoD	0.0121
Count of <h1> tag	0.0025
Count of <aside> tag	0.0008
defineProperty()	0.0024
Object.defineProperty()	0.0024
<img src=”*.png”>	0.0019
Math.max()	0.0029
Count of ‘k’ character	0.0070
<script src=”http*”>	0.0053
pragma no-cache	0.0123
<script language=”javascript”>	0.0045
Count of <input> tag	0.0011
connection close	0.0043
<form action> OoD	0.0016
get()	0.0013
<a href=”*.html”>	0.0077
forEach()	0.0029
Count of tag	4.4068
Count of <source> tag	0.0004
<a_href=”0*”>	0.0050
Count of ‘b’ character	0.0101
<a href=”*.pdf”>	0.0025
Count of <textarea> tag	0.0001
cache-control no-cache	0.0102
Count of tag	0.0022
keys()	0.0032
<img src> OoD	0.0070
slice()	0.0017
<link rel=”preload”>	0.0042
<meta http-equiv=”Content-Type“>	0.0234
Object.keys()	0.0032
<script language=”JavaScript”>	0.0045

Correlation Values between Target Variable and 288 Features on Dataset 2	
Features	***Dataset 2***
Count of 6-character words	0.0057
querySelector()	0.0035
<script crossorigin="anonymous">	0.0064
Count of 8-character words	0.0021
<a href="*.com">	0.0020
Count of 'p' character	0.0029
Count of <td> tag	0.0012
charAt()	0.0018
unescape()	0.0040
Count of 'g' character	0.0100
<iframe src="*0">	0.0025
Count of <dd> tag	0.0023
Count of <tbody> tag	0.0005
<form action="*.php">	0.0029
<form action="http*">	0.0044
script_charset_UTF-8	0.0051
<img src="http*">	0.0095
<img src="*.jpeg">	0.0007
toString()	0.0021
<script charset="utf-8">	0.0051
Count of <tr> tag	0.0017
Count of <base> tag	0.0035
<base href> absolute	0.0035
<base href> relative	0.0035
<a href="*.php">	0.0033
call()	0.0036
Count of 'l' character	0.0101
Count of <table> tag	0.0011
Count of <label> tag	0.0027
Count of 'j' character	0.0081
Count of <dt> tag	0.0033
Count of <font> tag	0.0002
Count of <fieldset> tag	0.0011
add()	0.0001
Count of tag	0.0007
<img src="*.gif">	0.0043
Math.round()	0.0032
<iframe src="http*">	0.0044
cache-control public	0.0024
Count of <title> tag	0.0058
<img src="*.svg">	0.0009
<iframe src="*.html">	0.0046
write()	0.0012

Table D-2:
Performance of a Several Models Built with Features from Prior Research Versus Discovered Features

Model Performance over Various Scenarios with 99 Prior Features / 34 Identified Features																				
Model	***No-sampling***							***Over-sampling***							***Under-sampling***						
	FPR	***FNR***	***ACC***	***AUC***	***MCC***	***Prec***	***Rec***	***FPR***	***FNR***	***ACC***	***AUC***	***MCC***	***Prec***	***Rec***	***FPR***	***FNR***	***ACC***	***AUC***	***MCC***	***Prec***	***Rec***
KNN	0.0050/ 0.0113	0.1844/ 0.1522	0.9743/ 0.9724	0.9052/ 0.9181	0.8686/ 0.8611	0.9544/ 0.9061	0.8155/ 0.8477	0.0608/ 0.0529	0.0744/ 0.0966	0.9376/ 0.9420	0.9323/ 0.9252	0.7517/ 0.7581	0.6637/ 0.6889	0.9255/ 0.9033	0.0474/ 0.0693	0.0955/ 0.0844	0.9470/ 0.9289	0.9285/ 0.9231	0.7741/ 0.7242	0.7121/ 0.6314	0.9044/ 0.9155
RF	0.0053/ 0.0061	0.1633/ 0.1300	0.9765/ 0.9795	0.9156/ 0.9319	0.8803/ 0.8968	0.9531/ 0.9479	0.8366/ 0.8700	0.0079/ 0.0099	0.1466/ 0.1233	0.9761/ 0.9770	0.9227/ 0.9333	0.8792/ 0.8850	0.9331/ 0.9195	0.8533/ 0.8766	0.0364/ 0.0425	0.0811/ 0.0666	0.9584/ 0.9547	0.9412/ 0.9454	0.8161/ 0.8070	0.7657/ 0.7400	0.9188/ 0.9333
AB	0.0129/ 0.0132	0.1555/ 0.1611	0.9706/ 0.9697	0.9157/ 0.9128	0.8525/ 0.8478	0.8941/ 0.8913	0.8444/ 0.8388	0.0351/ 0.0456	0.1188/ 0.0733	0.9552/ 0.9511	0.9229/ 0.9404	0.7959/ 0.7935	0.7647/ 0.7245	0.8811/ 0.9266	0.0622/ 0.0606	0.0633/ 0.0622	0.9376/ 0.9391	0.9371/ 0.9385	0.7553/ 0.7600	0.6611/ 0.6671	0.9366/ 0.9377
GB	0.0077/ 0.0080	0.1522/ 0.1511	0.9756/ 0.9755	0.9199/ 0.9204	0.8764/ 0.8758	0.9339/ 0.9317	0.8477/ 0.8488	0.0181/ 0.0298	0.1255/ 0.0900	0.9695/ 0.9632	0.9281/ 0.9400	0.8509/ 0.8319	0.8619/ 0.7982	0.8744/ 0.9100	0.0455/ 0.0458	0.0611/ 0.0588	0.9526/ 0.9526	0.9466/ 0.9476	0.8017/ 0.8023	0.7278/ 0.7270	0.9388/ 0.9411
ET	0.0047/ 0.0067	0.1844/ 0.1500	0.9746/ 0.9767	0.9053/ 0.9216	0.8699/ 0.8821	0.9569/ 0.9421	0.8155/ 0.8500	0.0070/ 0.0106	0.1488/ 0.1166	0.9766/ 0.9771	0.9220/ 0.9363	0.8815/ 0.8861	0.9398/ 0.9148	0.8511/ 0.8833	0.0371/ 0.0386	0.0788/ 0.0722	0.9580/ 0.9575	0.9419/ 0.9445	0.8153/ 0.8151	0.7626/ 0.7570	0.9211/ 0.9277
XGB	0.0079/ 0.0077	0.1522/ 0.1533	0.9755/ 0.9755	0.9199/ 0.9194	0.8757/ 0.8757	0.9327/ 0.9338	0.8477/ 0.8466	0.0181/ 0.0314	0.1277/ 0.0866	0.9692/ 0.9622	0.9270/ 0.9409	0.8495/ 0.8288	0.8616/ 0.7903	0.8722/ 0.9133	0.0451/ 0.0467	0.0655/ 0.0633	0.9525/ 0.9513	0.9446/ 0.9449	0.8001/ 0.7970	0.7287/ 0.7223	0.9344/ 0.9366
BC	0.0063/ 0.0083	0.1388/ 0.1433	0.9784/ 0.9761	0.9273/ 0.9241	0.8908/ 0.8793	0.9462/ 0.9300	0.8611/ 0.8566	0.0109/ 0.0132	0.1366/ 0.1311	0.9746/ 0.9732	0.9261/ 0.9278	0.8725/ 0.8666	0.9109/ 0.8947	0.8633/ 0.8688	0.0389/ 0.0446	0.0744/ 0.0777	0.9569/ 0.9515	0.9433/ 0.9387	0.8128/ 0.7935	0.7552/ 0.7280	0.9255/ 0.9222
NN	0.0136/ 0.0129	0.1300/ 0.1400	0.9729/ 0.9724	0.9273/ 0.9241	0.8655/ 0.8622	0.8918/ 0.8958	0.8700/ 0.8600	0.0220/ 0.0220	0.1100/ 0.1200	0.9678/ 0.9666	0.9261/ 0.9278	0.8463/ 0.8399	0.8396/ 0.8380	0.8900/ 0.8800	0.0563/ 0.0624	0.0755/ 0.0644	0.9414/ 0.9373	0.9433/ 0.9387	0.7626/ 0.7542	0.6802/ 0.6603	0.9244/ 0.9355
V	0.0028/ 0.0142	0.1266/ 0.0588	0.9831/ 0.9809	0.9352/ 0.9399	0.9135/ 0.9030	0.9747/ 0.9411	0.8733/ 0.8868	0.0066/ 0.0123	0.1018/ 0.1087	0.9826/ 0.9766	0.9457/ 0.9394	0.9117/ 0.8836	0.9452/ 0.9023	0.8981/ 0.8912	0.0378/ 0.0405	0.0463/ 0.0475	0.9612/ 0.9586	0.9578/ 0.9559	0.8321/ 0.8230	0.7622/ 0.7491	0.9536/ 0.9524

Table D-3:
Performance of a Several Models Built with Transformed Features from Prior Research Versus Discovered Features

Model Performance in Feature Transformation Scenarios with 99 Prior Features / 34 Identified Features														
Model	***Feature Transformation with Feature Selection***							***Feature Transformation with PCA***						
	FPR	***FNR***	***ACC***	***AUC***	***MCC***	***Prec***	***Rec***	***FPR***	***FNR***	***ACC***	***AUC***	***MCC***	***Prec***	***Rec***
KNN	0.0102/ 0.0113	0.1800/ 0.1655	0.9702/ 0.9709	0.9048/ 0.9115	0.8485/ 0.8527	0.9122/ 0.9048	0.8200/ 0.8344	0.0015/ 0.0102	0.2866/ 0.1788	0.9656/ 0.9703	0.8558/ 0.9054	0.8210/ 0.8492	0.9831/ 0.9123	0.7133/ 0.8211
RF	0.0063/ 0.0077	0.1566/ 0.1355	0.9763/ 0.9775	0.9184/ 0.9283	0.8799/ 0.8866	0.9452/ 0.9350	0.8433/ 0.8644	0.0057/ 0.0077	0.2811/ 0.1722	0.9626/ 0.9733	0.8565/ 0.9099	0.8039/ 0.8640	0.9417/ 0.9324	0.7188/ 0.8277
AB	0.0125/ 0.0142	0.2322/ 0.1666	0.9622/ 0.9682	0.8776/ 0.9095	0.8052/ 0.8402	0.8881/ 0.8833	0.7677/ 0.8333	0.0165/ 0.0149	0.1833/ 0.1711	0.9642/ 0.9670	0.9000/ 0.9069	0.8203/ 0.8344	0.8647 /0.8776	0.8166/ 0.8288
GB	0.0089/ 0.0070	0.1722/ 0.1566	0.9723/ 0.9757	0.9094/ 0.9181	0.8589/ 0.8768	0.9231/ 0.9393	0.8277/ 0.8433	0.0109/ 0.0096	0.1844/ 0.1644	0.9691/ 0.9725	0.9023/ 0.9129	0.8426/ 0.8607	0.9061/ 0.9181	0.8155/ 0.8355
ET	0.0080/ 0.0072	0.1688/ 0.1466	0.9734/ 0.9767	0.9115/ 0.9230	0.8648/ 0.8823	0.9303/ 0.9388	0.8311/ 0.8533	0.0011/ 0.0067	0.5822/ 0.2088	0.9321/ 0.9700	0.7083/ 0.8921	0.6153/ 0.8455	0.9791/ 0.9380	0.4177/ 0.7911
XGB	0.0089/ 0.0073	0.1700/ 0.1633	0.9725/ 0.9747	0.9105/ 0.9146	0.8603/ 0.8714	0.9233/ 0.9365	0.8300/ 0.8366	0.0093/ 0.0089	0.1755/ 0.1622	0.9715/ 0.9734	0.9075/ 0.9144	0.8550/ 0.8652	0.9194/ 0.9240	0.8244/ 0.8377
BC	0.0080/ 0.0086	0.1588/ 0.1333	0.9746/ 0.9770	0.9165/ 0.9290	0.8710/ 0.8843	0.9311/ 0.9285	0.8411/ 0.8666	0.0106/ 0.0082	0.1955/ 0.1611	0.9681/ 0.9742	0.8968/ 0.9153	0.8368/ 0.8690	0.9072/ 0.9298	0.8044/ 0.8388
NN	0.0131/ 0.0157	0.1511/ 0.1233	0.9710/ 0.9719	0.9165/ 0.9290	0.8547/ 0.8617	0.8935/ 0.8786	0.8488/ 0.8766	0.0129/ 0.0154	0.1222/ 0.1277	0.9744/ 0.9716	0.8968/ 0.9153	0.8733/ 0.8601	0.8977/ 0.8800	0.8777/ 0.8722
V	0.0057/ 0.0061	0.1544/ 0.1454	0.9774/ 0.9780	0.9198/ 0.9241	0.8837/ 0.8872	0.9493/ 0.9463	0.8455/ 0.8545	0.7152/ 0.1764	0.8489/ 0.1053	0.2696/ 0.8938	0.2179/ 0.5383	-0.375/ 0.2347	0.0262/ 0.8235	0.1510/ 0.0789

Table D-4:
Performance of a Random Forest Classifier Trained with 34 Features
on Dataset 3 Snapshot 1

Evaluation Over Time with a Random Forest Classifier Trained with 34 Features on Dataset 3, Snapshot 02/02/2020							
Snapshot	***ACC***	***FPR***	***FNR***	***AUC***	***MCC***	***Prec***	***Rec***
2/9/2020	0.9948	6.37E-05	0.1905	0.9046	0.8960	0.9971	0.8094
2/16/2020	0.9935	7.43E-05	0.2400	0.8799	0.8673	0.9964	0.7600
2/23/2020	0.9919	6.35E-05	0.2978	0.8510	0.8331	0.9967	0.7021
3/1/2020	0.9915	6.37E-05	0.3157	0.8420	0.8221	0.9966	0.6842
3/8/2020	0.9909	4.24E-05	0.3383	0.8307	0.8086	0.9976	0.6616
3/15/2020	0.9899	6.36E-05	0.3760	0.8119	0.7843	0.9962	0.6239
3/22/2020	0.9890	5.32E-05	0.4036	0.7981	0.7666	0.9967	0.5963
3/29/2020	0.9882	5.30E-05	0.4363	0.7817	0.7449	0.9966	0.5636
4/5/2020	0.9880	4.23E-05	0.4447	0.7776	0.7396	0.9972	0.5553
4/12/2020	0.9871	4.24E-05	0.4758	0.7620	0.7181	0.9970	0.5241
4/19/2020	0.9869	3.19E-05	0.4846	0.7576	0.7123	0.9977	0.5153

Table D-5:
Performance of a Random Forest Classifier Trained with 99 Features
on Dataset 3 Snapshot 1

Evaluation Over Time with a Random Forest Classifier Trained with 99 Features on Dataset 3, Snapshot 02/02/2020							
Snapshot	***ACC***	**FPR**	**FNR**	***AUC***	**MCC**	***Prec***	***Rec***
2/9/2020	0.9970	0	0.1123	0.9438	0.9407	1	0.8876
2/16/2020	0.9960	0	0.1491	0.9254	0.9205	1	0.8508
2/23/2020	0.9948	0	0.1932	0.9033	0.8958	1	0.8067
3/1/2020	0.9945	0	0.2036	0.8981	0.8899	1	0.7963
3/8/2020	0.9940	0	0.2238	0.8880	0.8782	1	0.7761
3/15/2020	0.9930	0	0.2614	0.8692	0.8563	1	0.7385
3/22/2020	0.9924	0	0.2820	0.8589	0.8440	1	0.7179
3/29/2020	0.9915	0	0.3155	0.8422	0.8237	1	0.6844
4/5/2020	0.9911	0	0.3306	0.8346	0.8144	1	0.6693
4/12/2020	0.9904	0	0.3546	0.8226	0.7994	1	0.6453
4/19/2020	0.9900	0	0.3695	0.8152	0.7900	1	0.6304

Table D-6:
Performance of a Random Forest Classifier Trained with Re-selected Features
on Dataset 3 Snapshot 1

Evaluation Over Time with a Random Forest Classifier Trained with Re-selected Features on Dataset 3, Snapshot 02/02/2020							
Snapshot	***ACC***	***FPR***	***FNR***	***AUC***	***MCC***	***Prec***	***Rec***
2/9/2020	0.9954	0.00010	0.1684	0.9157	0.9076	0.9953	0.8315
2/16/2020	0.9942	0.00011	0.2116	0.8941	0.8828	0.9946	0.7883
2/23/2020	0.9932	0.00012	0.2465	0.8766	0.8623	0.9938	0.7534
3/1/2020	0.9928	0.00012	0.2653	0.8672	0.8512	0.9937	0.7346
3/8/2020	0.9920	0.00013	0.2946	0.8525	0.8334	0.9929	0.7053
3/15/2020	0.9912	0.00013	0.3233	0.8382	0.8158	0.9926	0.6766
3/22/2020	0.9905	8.51E-05	0.3470	0.8264	0.8022	0.9953	0.6529
3/29/2020	0.9897	0.00011	0.3750	0.8124	0.7837	0.9932	0.6250
4/5/2020	0.9893	9.52E-05	0.3940	0.8029	0.7719	0.9943	0.6059
4/12/2020	0.9884	0.00014	0.4256	0.7871	0.7498	0.9907	0.5743
4/19/2020	0.9879	0.00012	0.4422	0.7788	0.7391	0.9917	0.5577

Table D-7:
Performance of a Random Forest Classifier Trained with 34 Features
on Dataset 3 Snapshot 6

Evaluation Over Time with a Random Forest Classifier Trained with 34 Features on Dataset 3, Snapshot 03/08/2020							
Snapshot	***ACC***	***FPR***	***FNR***	***AUC***	***MCC***	***Prec***	***Rec***
3/15/2020	0.9949	4.24E-05	0.1893	0.9052	0.8971	0.9980	0.8106
3/22/2020	0.9930	2.13E-05	0.2589	0.8705	0.8573	0.9989	0.7410
3/29/2020	0.9914	5.30E-05	0.3148	0.8425	0.8229	0.9972	0.6851
4/5/2020	0.9905	3.17E-05	0.3517	0.8241	0.8005	0.9982	0.6482
4/12/2020	0.9898	3.18E-05	0.3769	0.8115	0.7845	0.9981	0.6230
4/19/2020	0.9893	4.25E-05	0.3941	0.8029	0.7731	0.9974	0.6058

Table D-8:
Performance of a Random Forest Classifier Trained with 99 Features
on Dataset 3 Snapshot 6

Evaluation Over Time with a Random Forest Classifier Trained with 99 Features on Dataset 3, Snapshot 03/08/2020							
Snapshot	***ACC***	***FPR***	***FNR***	***AUC***	***MCC***	***Prec***	***Rec***
3/15/2020	0.9970	0	0.1119	0.9440	0.9409	1	0.8880
3/22/2020	0.9954	0	0.1689	0.9155	0.9095	1	0.8310
3/29/2020	0.9945	0	0.2032	0.8983	0.8901	1	0.7967
4/5/2020	0.9939	0	0.2269	0.8865	0.8764	1	0.7730
4/12/2020	0.9930	0	0.2588	0.8705	0.8578	1	0.7411
4/19/2020	0.9925	0	0.2767	0.8616	0.8472	1	0.7232

Table D-9:
Performance of a Random Forest Classifier Trained with Re-selected Features
on Dataset 3 Snapshot 6

Evaluation Over Time with a Random Forest Classifier Trained with Re-selected Features on Dataset 3, Snapshot 03/08/2020							
Snapshot	***ACC***	***FPR***	***FNR***	***AUC***	***MCC***	***Prec***	***Rec***
3/15/2020	0.9953	7.42E-05	0.1731	0.9134	0.9056	0.9967	0.8268
3/22/2020	0.9939	6.39E-05	0.2216	0.8891	0.8782	0.9970	0.7783
3/29/2020	0.9927	7.42E-05	0.2676	0.8661	0.8510	0.9963	0.7323
4/5/2020	0.9919	6.34E-05	0.2972	0.8513	0.8334	0.9967	0.7027
4/12/2020	0.9911	0.0001	0.3236	0.8381	0.8161	0.9938	0.6763
4/19/2020	0.9905	9.57E-05	0.3470	0.8259	0.8014	0.9947	0.6520

Table D-10:
Performance of a Random Forest Classifier Trained with 34 Features
on Dataset 3 Snapshot 1-6

Evaluation Over Time with a Random Forest Classifier Trained with 34 Features on Dataset 3, Snapshot 02/02/2020-03/08/2020							
Snapshot	***ACC***	***FPR***	***FNR***	***AUC***	***MCC***	***Prec***	***Rec***
3/15/2020	0.9982	0.00015	0.0584	0.9706	0.9664	0.9938	0.9415
3/22/2020	0.9974	0.00012	0.0908	0.9545	0.9498	0.9949	0.9091
3/29/2020	0.9965	0.00013	0.1230	0.9383	0.9321	0.9943	0.8769
4/5/2020	0.9960	0.00015	0.1432	0.9283	0.9206	0.9933	0.8567
4/12/2020	0.9952	0.00013	0.1710	0.9144	0.9055	0.9940	0.8289
4/19/2020	0.9949	0.00014	0.1839	0.9079	0.8979	0.9934	0.8160

Table D-11:
Performance of a Random Forest Classifier Trained with 99 Features
on Dataset 3 Snapshot 1-6

Evaluation Over Time with a Random Forest Classifier Trained with 99 Features on Dataset 3, Snapshot 02/02/2020-03/08/2020							
Snapshot	***ACC***	***FPR***	***FNR***	***AUC***	***MCC***	***Prec***	***Rec***
3/15/2020	0.9994	0	0.0224	0.9887	0.9884	1	0.9775
3/22/2020	0.9988	0	0.0430	0.9784	0.9776	1	0.9569

Evaluation Over Time with a Random Forest Classifier Trained with 99 Features on Dataset 3, Snapshot 02/02/2020-03/08/2020							
Snapshot	***ACC***	***FPR***	***FNR***	***AUC***	***MCC***	***Prec***	***Rec***
3/29/2020	0.9985	0	0.0552	0.9723	0.9712	1	0.9447
4/5/2020	0.9981	0	0.0675	0.9662	0.9647	1	0.9324
4/12/2020	0.9976	0	0.0878	0.9560	0.9539	1	0.9121
4/19/2020	0.9973	0	0.0981	0.9509	0.9483	1	0.9018

Table D-12:
Performance of a Random Forest Classifier Trained with Re-selected Features
on Dataset 3 Snapshot 1-6

Evaluation Over Time with a Random Forest Classifier Trained with Re-selected Features on Dataset 3, Snapshot 02/02/2020-03/08/2020							
Snapshot	***ACC***	***FPR***	***FNR***	***AUC***	***MCC***	***Prec***	***Rec***
3/15/2020	0.9978	0.00012	0.0755	0.9621	0.9580	0.9949	0.9244
3/22/2020	0.9971	0.00014	0.0988	0.9504	0.9450	0.9940	0.9011
3/29/2020	0.9966	0.00014	0.1207	0.9395	0.9331	0.9939	0.8792
4/5/2020	0.9962	0.00015	0.1347	0.9325	0.9252	0.9933	0.8652
4/12/2020	0.9956	0.00018	0.1552	0.9222	0.9134	0.9923	0.8447
4/19/2020	0.9953	0.00014	0.1685	0.9156	0.9066	0.9935	0.8314

Table D-13:
Details of Which Features Changed Over Time, Beginning with the First Snapshot

Rejecting the Null Hypothesis (1 = Reject, 0 = Cannot Reject) for Features (Related T Test / KS / k-sample Anderson-Darling / Kruskal Wallis H-test											
Feature	***2/2/20 - 2/9/20***	***2/2/20 - 2/16/20***	***2/2/20 - 2/23/20***	***2/2/20 - 3/1/20***	***2/2/20 - 3/8/20***	***2/2/20 - 3/15/20***	***2/2/20 - 3/22/20***	***2/2/20 - 3/29/20***	***2/2/20 - 4/5/20***	***2/2/20 - 4/12/20***	***2/2020 - 4/19/20***
<a href="http*">	0/0/0/0	0/0/0/0	0/1/1/1	0/1/1/1	0/1/1/1	0/1/1/1	1/1/1/1	1/1/1/1	1/1/1/1	1/1/1/1	1/1/1/1
<a href="https*">	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/1	0/1/1/1	0/1/1/1	1/1/1/1	1/1/1/1	1/1/1/1
Count of tag	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/1/0
Count of tag	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/1/0
createElement()	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/1/0
write()	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/1/1	0/0/1/1
addEventListener()	0/0/0/0	0/0/0/0	0/0/1/0	1/0/1/1	1/0/1/1	1/0/1/1	1/0/1/1	1/0/1/1	1/0/1/1	1/0/1/1	1/1/1/1
<form action="*.php">	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	1/0/1/1	1/0/0/0	1/0/0/0	1/0/0/0	1/0/1/1
<form action="http*">	0/0/0/0	0/0/0/0	0/0/1/1	0/0/1/0	0/0/1/1	0/0/1/1	0/0/1/1	0/0/1/1	0/0/1/1	0/0/1/1	0/0/1/1
cache-control max-age	0/0/0/0	0/0/1/0	0/0/0/0	0/0/1/0	0/0/1/1	1/0/1/1	1/0/1/1	1/0/1/1	1/0/1/1	1/0/1/1	1/1/1/1
cache-control must-revalidate	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/1/0
cache-control no-cache	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/1/0	0/0/1/1
cache-control no-store	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/1/0	0/0/0/0	0/0/0/0	1/0/1/1	1/1/1/1
cache-control public	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/1/0	1/0/1/1
content-encoding gzip	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	1/0/1/1
content-language text/html	0/0/1/1	0/0/1/1	1/0/1/1	1/0/0/0	0/0/1/1	0/0/0/0	0/0/1/1	0/0/1/1	0/0/0/0	0/0/1/1	0/0/1/1
content-length	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/1/1	0/0/1/0	0/0/1/0	0/1/1/1	0/1/1/1
expect-ctreport-uri	0/0/0/0	0/0/0/0	0/0/0/0	0/0/1/0	1/0/1/1	1/0/1/1	1/0/1/1	1/0/1/1	1/1/1/1	1/1/1/1	1/1/1/1
server apache	0/0/0/0	0/0/0/0	0/0/0/0	0/0/1/1	0/0/1/1	1/1/1/1	1/1/1/1	1/1/1/1	1/1/1/1	1/1/1/1	1/1/1/1
strict-transport-security_max-age	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	1/0/1/0	1/0/1/1	1/0/1/1	1/1/1/1	1/1/1/1	1/1/1/1	1/1/1/1
transfer-encoding chunked	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/1/1	0/0/1/0	1/0/1/0	1/0/1/1	1/1/1/1
via_1.1	0/0/0/0	0/0/0/0	0/0/0/0	1/0/1/1	1/0/1/1	1/0/1/1	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0
hidden <iframe>	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/1/0
<img src="*.jpg">	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/1/1	0/0/0/0	0/0/1/1	0/1/1/1	0/1/1/1	0/1/1/1
<img src="http*">	0/0/0/0	0/0/0/0	0/0/1/1	0/0/1/1	0/0/1/1	0/0/1/1	0/1/1/1	0/1/1/1	0/1/1/1	0/1/1/1	1/1/1/1
Count of <input> tag	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/1	0/0/0/1	0/1/1/1	0/1/1/1
charAt()	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	1/0/0/0	1/0/0/0	1/0/1/0	1/0/1/0	1/0/0/0
charCodeAt()	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	1/0/0/0	1/0/0/0	1/0/0/0	1/0/1/0	1/0/0/0
push()	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/1/1	0/0/1/1	0/0/1/0
search()	0/0/0/0	0/0/1/1	0/0/0/0	0/0/1/1	0/0/0/0	0/0/0/0	0/0/1/1	0/0/0/0	0/0/0/0	0/0/0/0	0/0/1/1
shift()	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/1/0	0/0/1/0
escape()	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/1/0	0/0/1/1	0/0/1/1
eval()	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/1/1	0/0/1/1	0/0/1/1	0/0/1/1	0/0/1/1	0/0/1/1	0/0/0/0
unescape()	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/1/0	0/0/0/0
Count of <link> tag	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	1/0/1/1	1/0/1/0	1/0/1/0	1/0/1/0	1/1/1/1

Rejecting the Null Hypothesis (1 = Reject, 0 = Cannot Reject) for Features (Related T Test / KS / k-sample Anderson-Darling / Kruskal Wallis H-test											
Feature	***2/2/20 - 2/9/20***	***2/2/20 - 2/16/20***	***2/2/20 - 2/23/20***	***2/2/20 - 3/1/20***	***2/2/20 - 3/8/20***	***2/2/20 - 3/15/20***	***2/2/20 - 3/22/20***	***2/2/20 - 3/29/20***	***2/2/20 - 4/5/20***	***2/2/20 - 4/12/20***	***2/2020 - 4/19/20***
<link href="*.php">	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/1/0	0/0/1/1	0/0/1/1	0/0/1/1
<link href="https*">	0/0/0/0	0/0/0/0	0/0/0/0	0/0/1/1	0/0/1/1	1/1/1/1	1/1/1/1	1/1/1/1	1/1/1/1	1/1/1/1	1/1/1/1
<link href> relative	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/1/0/0	0/1/1/1	0/1/0/0	0/1/0/0	1/0/0/0	1/0/0/1
<link type="text/css">	0/0/0/0	0/0/0/0	0/0/1/0	0/0/0/0	0/0/1/1	0/0/1/1	0/0/1/1	0/1/1/1	0/1/1/1	0/1/1/1	0/1/1/1
Meta content index follow	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/1/0	0/0/1/1
Count of <meta> tag	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/1/0
getElementsByTagName()	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/1/1	0/0/1/1	1/1/1/1	1/0/1/1
<script src="https*">	0/0/0/0	0/0/0/0	0/0/0/0	0/0/1/1	0/1/1/1	0/1/1/1	0/1/1/1	0/1/1/1	1/1/1/1	1/1/1/1	1/1/1/1
<script type=text/javascript>	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/1/1	0/0/1/1	1/1/1/1	1/1/1/1	1/1/1/1	1/1/1/1
<source srcset> OoD	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/1/0	0/0/1/1	0/0/1/1	0/0/1/1	0/0/1/1
Count of <style> tag	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	1/0/1/1	1/0/1/1	1/0/1/1	1/1/1/1	1/1/1/1	1/1/1/1
Total HTML Tags	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/1/0/0	0/1/0/0
Total <img src>	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/0/0	0/0/1/1	0/0/1/1	0/0/1/0

Table D-14:
Feature Change Based on the Related T-test

Number of Features That Change Per a Given Time Difference (Measuring All Possible Intervals) - Related T-test											
Time Difference (Weeks)	***Measurements***										
1	0	0	1	1	1	0	1	0	0	0	0
2	0	1	2	2	1	1	1	0	0	0	
3	1	2	1	3	3	3	2	0	0		
4	3	1	2	4	7	5	6	3			
5	4	4	4	6	10	8	7				
6	8	10	9	9	12	10					
7	12	13	11	11	15						
8	13	15	13	15							
9	16	17	17								
10	19	22									
11	22										

Table D-15:
Feature Change Based on the Two-Sample KS Test

Number of Features That Change Per a Given Time Difference (Measuring All Possible Intervals) – Kolmogorov Smirnov											
Time Difference (Weeks)	***Measurements***										
1	0	0	0	0	0	1	0	0	0	0	0
2	0	0	0	0	1	1	0	0	0	0	
3	1	1	0	2	1	2	0	1	0		
4	1	1	3	2	2	3	1	1			
5	2	3	3	4	4	6	3				
6	5	5	7	6	7	6					
7	7	6	12	10	10						
8	10	12	18	11							
9	13	19	15								
10	16	15									
11	20										

Table D-16:
Feature Change Based on the K-Sample Anderson-Darling

Number of Features That Change Per a Given Time Difference (Measuring All Possible Intervals) - k-sample Anderson Darling											
Time Difference (Weeks)	***Measurements***										
1	1	0	3	4	2	1	3	4	0	1	5
2	3	5	3	4	4	5	2	7	1	9	
3	6	5	4	6	6	9	5	11	12		
4	11	6	9	17	12	14	11	9			
5	14	13	13	13	15	21	23				
6	16	20	20	17	21	25					
7	24	24	28	25	28						
8	23	29	31	32							
9	27	34	36								
10	36	40									
11	41										

Table D-17:
Feature Change Based on the Kruskal Wallis H Test

Number of Features That Change Per a Given Time Difference (Measuring All Possible Intervals) - Kruskal Wallis H test											
Time Difference (Weeks)	***Measurements***										
1	1	0	2	3	2	1	3	4	0	0	4
2	2	4	2	4	4	2	1	4	0	8	
3	4	5	4	5	5	5	4	8	7		
4	8	5	10	11	12	11	9	6			
5	13	12	10	8	13	18	15				
6	17	17	17	15	18	21					

Number of Features That Change Per a Given Time Difference (Measuring All Possible Intervals) - Kruskal Wallis H test											
Time Difference (Weeks)	***Measurements***										
7	22	18	21	22	21						
8	20	21	25	29							
9	22	28	26								
10	28	30									
11	33										

References

1. Pew Research Center, “Internet/broadband fact sheet,” Washington D.C., June 12, 2019. Available: https://www.pewresearch.org/internet/fact-sheet/internet-broadband/

2. Internet World Stats, “Internet growth statistics.” Available: https://www.internetworldstats.com/emarketing.htm [Accessed: September 19, 2019].

3. Digital Commerce 360, “U.S. e-commerce sales grow 15.0%.” Available: https://www.digitalcommerce360.com/article/us-ecommerce-sales/. [Accessed: Sept. 19, 2019].

4. H. Cavusoglu, B. Mishra, and S. Raghunathan, “The effect of internet security breach announcements on market value: Capital market reactions for breached firms and internet security developers,” *International Journal of Electronic Commerce*, vol. 9, no. 1, pp. 69–104, Fall 2004.

5. S. Morgan, “Global cybersecurity spending predicted to exceed $1 trillion from 2017-2021,” *Cybercrime Magazine*, June 10, 2019. Available: https://cybersecurityventures.com/cybersecurity-market-report/. [Accessed: Sept. 19, 2019].

6. R. Tonar and E. Talton, “A lack of cybersecurity funding and expertise threatens U.S. infrastructure,” *Forbes*, April 23, 2018. Available: https://www.forbes.com/sites/ellistalton/2018/04/23/the-u-s-governments-lack-of-cybersecurity-expertise-threatens-our-infrastructure. [Accessed: Sept. 19, 2019].

7. L. J. Trautman and P. C. Ormerod, “Corporate directors' and officers' cybersecurity standard of care: The Yahoo data breach,” *66 American University Law Review*, vol 66, no. 1231, 2017.

8. Broadcom.com, “Hackers attack Epsilon database, phishing spree anticipated,” April 13, 2011. Available: https://www.symantec.com/connect/blogs/hackers-attack-epsilon-database-phishing-spree-anticipated. [Accessed: Sept. 19, 2019].

9. M. Antonakakis, et al., “Understanding the Mirai botnet,” in *Proc. of the 6th USENIX Security Symposium*, 2017, pp. 1093–1110.

10. S. Weagle, “Financial impact of Mirai DDos attack on Dyn revealed in new data,” *The DDoS Blog*, Feb. 21, 2017, Available: https://www.corero.com/blog/797-financial-impact-of-mirai-ddos-attack-on-dyn-revealed-in-new-data.html. [Accessed: Sept. 19, 2019].

11. T. N. Jagatic, N. A. Johnson, M. Jakobsson, and F. Menczer, “Social phishing,” *Communications of the ACM*, vol. 50, no. 10, pp. 94–100, October 2007.

12. V. L Le, I. Welch, X. Gao, and P. Komisarczuk, “Anatomy of drive-by download attack,” in *Proc. of the 11th Australasian Information Security Conference-Volume,* 2013, pp. 49-58.

13. J. Mirkovic and P. Reiher, “A taxonomy of DDoS attack and DDoS defense mechanisms,” *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 2, pp.39–53, 2004.

14. W3Schools.com, “JavaScript Tutorial.” Available: https://www.w3schools.com/js/. [Accessed: Sept. 19, 2019].

15. J. Gardiner, M. Cova, and S. Nagaraja, “Command and control: Understanding, denying and detecting, a review of malware C2 techniques, detection and defences.” *ArXiv preprint arXiv:1408.1136.*, 38 pages, 2014.

16. J. Vijayan, “More than 22,000 vulns were disclosed in 2018, 27% without fixes,” *DarkReading.com,* Feb. 27, 2019. Available: https://www.darkreading.com/vulnerabilities---threats/more-than-22000-vulns-were-disclosed-in-2018-27--without-fixes/d/d-id/1333998. [Accessed: Sept. 19, 2019].

17. H. Orman, “The Morris Worm: A fifteen-year perspective,” *IEEE Security & Privacy*, vol. 1, no. 5, pp.35–43, Sept./Oct. 2003.

18. C. Willems, T. Holz, and F. Freiling, “Toward automated dynamic malware analysis using cwsandbox,” *IEEE Security & Privacy*, vol. 5, no. 2, pp.32–39, March/April 2007.

19. A. Malanov, “Antivirus fundamentals: Viruses, signatures, disinfection,” *Kaspersky Daily*, Oct. 13, 2016. Available: https://www.kaspersky.com/blog/signature-virus-disinfection/13233/. [Accessed: Sept. 19, 2019].

20. Verisign, “What is a URL?” Available: https://www.verisign.com/en_US/website-presence/online/what-is-a-url/index.xhtml. [Accessed: Sept. 19, 2019].

21. Mozilla MDN Web Docs, “HTML: Hypertext markup language,” updated July 27, 2020. Available: https://developer.mozilla.org/en-US/docs/Web/HTML. [Accessed: Sept. 19, 2019].

22. Pixabay. “Stunning free images and royalty free stock.” Available: https://pixabay.com/. [Accessed: Sept. 19, 2019].

23. C. Seifert, I. Welch, and P. Komisarczuk, “Identification of malicious web pages with static heuristics,” in *Proc. of the 2008 Australasian Telecommunication Networks and Applications Conference, Adelaide, SA, Australia, Dec. 7–10, 2008,* pp. 91–96.

24. Y. Zhang, J. Hong, and L. Cranor, “CANTINA: A content-based approach to detecting phishing web sites," in *Proc. of the 16th International Conference on the World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8–12, 2007*, pp. 639–648.

25. S. B. Kotsiantis, I. Zaharakis, and P. Pintelas, “Supervised machine learning: A review of classification techniques,” *Informatica*, vol. 31, pp.249–268, January 2007.

26. R. Kohavi, “A study of cross-validation and bootstrap for accuracy estimation and model selection,” in *IJCAI ’95: Proc. of the 14th International Joint Conference on Artificial Intelligence*, San Francisco, CA: Morgan Kaufman Publishers, 1995, pp. 1137–1145.

27. S. Boughorbel, F. Jarray, and M. El-Anbari, “Optimal classifier for imbalanced data using Matthews Correlation Coefficient metric,” *PloS ONE*, vol. 12, no.6, June 2, 2017, p. e0177678.

28. S. Wold, K. Esbensen, and P. Geladi, “Principal component analysis,” *Chemometrics and Intelligent Laboratory Systems*, vol. 2, no 1–3, pp. 37–52, August 1987.

29. Scikit-Learn, 2019. Available: http://scikitlearn.org/. [Accessed: Feb. 8, 2019].

30. J. McGahagan IV, D. Bhansali, M. Gratian, and M. Cukier, “A comprehensive evaluation of HTTP headers for detecting malicious websites,” in *Proc. of the 2019 15th European Dependable Computing Conference (EDCC), Naples, Italy, Sept. 17–20, 2019*, pp. 75–82.

31. J. McGahagan IV, D. Bhansali, C. Pinto-Coelho, and M. Cukier, “A comprehensive evaluation of webpage content features for detecting malicious websites,” in *Proc. of the 2019 9th Latin America Symposium on Dependable Computing, Natal, Brazil, Nov. 19–21, 2019*, pp. 1–10.

32. Network Solutions, “Tools and Tips” Available: https://www.networksolutions.com/support/what-is-a-domain-name-server-dns-and-how-does-it-work/. [Accessed: Sept. 19. 2019].

33. Technopedia, “Website security certificate,” updated April 14, 2014. Available: https://www.techopedia.com/definition/29743/website-security-certificate/. [Accessed: Sept. 19, 2019].

34. S. Garera, N. Provos, M. Chew, and A. D. Rubin, “A framework for detection and measurement of phishing attacks,” in *WORM ’07: Proc. of the 2007 ACM Workshop on Recurring Malcode*. *Alexandria, VA, Nov. 2, 2007*, pp. 1–8.

35. J. Ma, L. K. Saul, S. Savage. and G. M. Voelker, “Beyond blacklists: Learning to detect malicious web sites from suspicious URLs,” in *Proc. of the 15th ACM*

SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28–July 1, 2009, pp. 1245–1254.

36. J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Identifying suspicious URLs: An application of large-scale online learning," in *ICML '09: Proc. of the 26th International Conference on Machine Learning, Montreal, Quebec, Canada, June 2009*, pp. 681–688.

37. J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Learning to detect malicious urls," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no.3, May 2011.

38. A. Blum, B. Wardman, T. Solorio, and G. Warner, "Lexical feature-based phishing URL detection using online learning," in *Proc. of the 3rd ACM Workshop on Security and Artificial Intelligence, AlSec 2010, Chicago, IL, Oct. 8, 2010*, pp. 54–60.

39. A. Le, A. Markopoulou, and M. Faloutsos, "Phishdef: URL names say it all," in *Proc. of the 2011 IEEE INFOCOM, Shanghai, China, April 10–15, 2011*, pp. 191–195.

40. A. E. Kosba, A. Mohaisen, A. West, T. Tonn, and H. K. Kim, "ADAM: Automated detection and attribution of malicious webpages," in *Proc. of the 2013 IEEE Conference on Communications and Security, National Harbor, MD, Oct. 14–16, 2013*, pp. 399–400.

41. T. Fisher, "What is an IP address?" *Lifewire*, March 9, 2020. Available: https://www.lifewire.com/what-is-an-ip-address-2625920. [Accessed: Sept. 19, 2019].

42. C. Whittaker, B. Ryner, and M. Nazif, "Large-scale automatic classification of phishing pages," in *Proc. of the Network and Distributed System Security Symposium, NDSS 2010, San Diego, California, Feb. 28-March 3, 2010*. Available: https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/35580.pdf.

43. M. He et al., "An efficient phishing webpage detector," *Expert Systems with Applications*, vol. 38, no. 10, pp. 12018–12027, September 2011.

44. S. Gastellier-Prevost, G. G. Granadillo, and M. Laurent, M, "Decisive heuristics to differentiate legitimate from phishing sites," in *Proc. of the 2011 Conference on Network and Information Systems Security, La Rochelle, France, May 18-21, 2011*, pp. 1-9.

45. D. Canali, M. Cova, G. Vigna, and C. Kruegel, "Prophiler: A fast filter for the large-scale detection of malicious web pages," in *Proc. of the 20th World Wide Web Conference, Hiderabad, India, March 2010*, pp. 197-206.

46. G. Xiang, J. Hong, C. P. Rose, and L. Cranor, "CANTINA+: A feature-rich machine learning framework for detecting phishing web sites," *ACM Transactions on Information and System Security,* vol. 14, no. 2, pp. 21, September 2011.

47. B. Eshete, A. Villafiorita, and K. Weldemariam, "BINSPECT: Holistic analysis and detection of malicious web pages," in *Proc. of the International Conference on Security and Privacy in Communication Systems, SecureComm 2012, Padua, Italy, Sept. 3–5, 2012*, pp. 149–166.

48. X. Gu, W. Hongyuan, and N. I. Tongguang, "An efficient approach to detecting phishing web," *Journal of Computational Information Systems*, vol. 9, no. 14, pp. 5553–5560, July 2013.

49. L. Xu, Z. Zhan, S. Xu, and K. Ye, "Cross-layer detection of malicious websites," in *Proc. of the Third ACM conference on Data and Application Security and Privacy. February 2013,* pp. 141–152.

50. R. B. Basnet, A. H. Sung, and Q. Liu, "Feature selection for improved phishing detection," in *Advanced Research in Applied Artificial Intelligence, IEA/AIE 2010*, vol. 7345, H. Jiang, W. Ding, M. Ali, and X. Wu, Eds. Berlin: Springer, 2012, pp. 252–261.

51. M. S. Lin, C. Y. Chiu, Y. J. Lee, and H. K. Pao, "Malicious URL filtering: A big data application," in *Proc. of the 2013 IEEE International Conference on Big Data, Silicon Valley, CA, Oct. 6–9, 2013*, pp. 589–596.

52. A. Ahluwalia, I. Traore, K. Ganame, and N. Agarwal, "Detecting broad length algorithmically generated domains," in Traore I., Woungang I., Awad A. (eds.) *Intelligent, Secure, and Dependable Systems in Distributed and Cloud Environments: ISDDC 2017,* I. Traore, I. Woungang, and A. Awad, Eds., Cham, Switzerland: Springer, 2017, pp. 19–34.

53. P. Arntz, "Explained: Domain-generating algorithms," *Malwarebytes Labs*, Dec. 6, 2016. [Online]. Available: https://blog.malwarebytes.com/security-world/2016/12/explained-domain-generating-algorithm/. [Accessed: Sept. 19, 2019].

54. H. Pao, Y. Chou, and Y. Lee, "Malicious URL detection based on Kolmogorov complexity estimation," in *Proc. of the 2012 IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technology, Macau, China, Dec. 4–7, 2012*, pp. 380–387.

55. N. Kheir, G. Blanc, H. Debar, J. Garcia-Alfaro, and D. Yang, "Automated classification of C&C connections through malware URL clustering," in *ICT Systems Security and Privacy Protection. SEC 2015. IFIP Advances in Information and Communication Technology*, H. Federrath and D. Gollman, Eds. Sham, Switzerland: Springer, 2015, pp. 252–266.

56. R. Verma and A. Das, “What's in a URL: Fast feature extraction and malicious URL detection,” in *IWSPA ’17: Proc. of the 3rd ACM on International Workshop on Security and Privacy Analytics, Scottsdale, AZ, March 2017*, pp. 55–63.

57. Tfidf.com, “What does TF-IDF mean?” [Online]. Available: http://www.tfidf.com/. [Accessed: May 23, 2019].

58. N. Sanglerdsinlapachai and A. Rungsawang, “Using domain top-page similarity feature in machine learning-based web phishing detection,” in *2010 Third International Conference on Knowledge Discovery and Data Mining, Phuket, Thailand, Jan. 9–10, 2010,* pp. 187–190.

59. N. Provos, P. Mavrommatis, M. Rajab, and F. Monrose, “All your iframes point to us,” in *SS '08: Proc. of the 17th Conference on Security Symposium,* Berkeley, CA: USENIX Association, 2008, pp. 1–15.

60. J. Drew, and T. Moore, “Automatic identification of replicated criminal websites using combined clustering,” in *2014 IEEE Security and Privacy Workshops, San Jose, CA, May 17–18, 2014*, pp. 116–123.

61. I. Corona et al., “Deltaphish: Detecting phishing webpages in compromised websites,” in *Computer Security – ESORICS 2017: 22nd European Symposium on Research in Computer Security, Oslo, Norway, Sept. 11-15, 2017, Proceedings, Part I*, pp. 370–388.

62. K. Borgolte, C. Kruegel, and G. Vigna, “Delta: Automatic identification of unknown web-based infection campaigns,” in *Proc. of the 2013 ACM SIGSAC Conference on Computer & Communications Security, Berlin, Germany, Nov. 4–8, 2013*, pp.109–120.

63. V. L. Le, I. Welch, X. Gao, and P. Komisarczuk, “Identification of potential malicious web pages,” in *AISC ’11: Proc. of the Ninth Australasian Information Security Conference, vol. 16,* Darlinghurst, NSW, Australia: Australian Computer Society Inc., 2011, pp. 33–40.

64. R. B. Basnet and A. H. Sung, “Learning to detect phishing webpages,” *Journal of Internet Service and Information. Security*, vol. 4, no. 3, pp. 21–39, August 2014.

65. M. Cova, C. Kruegel, and G. Vigna, “Detection and analysis of drive-by-download attacks and malicious JavaScript code,” in *WWW ’10: Proc. of the 19th International Conference on World Wide Web, Raleigh, NC, April 26–30, 2010,* pp. 281–290.

66. G. Canfora, and A. V. Corrado, “A set of features to detect web security threats,” *Journal of Computer Virology and Hacking Techniques,* vol. 12, no. 4, pp 243-261, January 2016.

67. K. Rieck, T. Krueger, and A. Dewald, “Cujo: Efficient detection and prevention of drive-by download attacks,” in *Proc. of the 26th Annual Computer Security Applications Conference, ACSAC 2010, Austin, TX, Dec. 6–10, 2010,* pp. 31–39.

68. C. Curtsinger, B. Livshits, B. G. Zorn, and C. Seifert, “ZOZZLE: Fast and precise in-browser JavaScript malware detection,” in *Proc. of the 20th USENIX Security Symposium, San Francisco, CA, Aug. 10–11, 2011,* pp. 33–48.

69. W. Xu, F. Zhang, and S. Zhu, “JStill: Mostly static detection of obfuscated malicious JavaScript code,” in *Proc. of the 3rd ACM Conference on Data and Application Security and Privacy, CODASPY’13, San Antonio, TX, Feb. 18–20, 2013,* pp. 117–128.

70. Mozilla MDN Web Docs, “Introduction to the DOM.” [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction. [Accessed: Sept. 19, 2019].

71. A. Kapravelos, Y. Shoshitaishvili, M. Cova, C. Kruegel, and G. Vigna, “Revolver: An automated approach to the detection of evasive web-based malware,” in *Proceedings of the 22nd USENIX Security Symposium, Washington DC, Aug. 14–16, 2013,* pp. 637–652.

72. H. Choi, B. B. Zhu, and H. Lee, “Detecting malicious web links and identifying their attack types,” in *WebApps ’11:Proceedings of the 2nd USENIX Conference on Web Application Development,* Berkeley, CA: USENIX Association, 2011, p. 218–229.

73. M. Heiderich, T. Frosch, and T. Holz, “Iceshield: Detection and mitigation of malicious websites with a frozen DOM,” in *Recent Advances in Intrusion Detection, RAID 2011*, R. Sommer, D. Balzarotti, and G. Maier, Eds., Berlin: Springer, 2011, pp. 281–300.

74. V. Beal, “HTTP-Hypertext transfer protocol,” *Webopedia*. [Online]. Available: https://www.webopedia.com/TERM/H/HTTP.html. [Accessed: Sept. 19, 2019].

75. R. Perdisci, W. Lee, and N. Feamster, “Behavioral clustering of http-based malware and signature generation using malicious network traces,” in *Proc. of the 7th USENIX conference on Networked Systems Design and Implementation, NSDI ‘10, San Jose, CA, April 28–30, 2010,* pp. 26–40.

76. R. Perdisci, A. Davide, and G. Giacinto, “Scalable fine-grained behavioral clustering of HTTP-based malware,” *Computer Networks,* vol. 57, no. 2, pp 487–500, February 2013.

77. W. Tao, Y. Shunzheng, and X. Bailin, “A novel framework for learning to detect malicious web pages,” *Proc. of the 2010 International Forum on Information*

Technology and Applications, IFITA 2010, Kunming, China, July 16–18, 2010, vol. 2, pp. 353–357.

78. J. Zhang, C. Seifert, J. W. Stokes, and W. Lee, "Arrow: Generating signatures to detect drive-by downloads," in *Proc. of the 20th International Conference on World Wide Web, WWW 2011, Hyderabad, India, March 2011*, pp. 187–196.

79. F. Brezo, J. G. de la Puerta, X. Ugarte-Pedrero, I. Santos, and P. G. Bringas, "A supervised classification approach for detecting packets originated in a HTTP-based botnet," *CLEI Electronic Journal*, vol. 16, no.3, pp. 2, December 2013.

80. T. Nelms, R. Perdisci, and M. Ahamad, "ExecScent: Mining for new C&C domains in live networks with adaptive control protocol templates," in *Proc. of the 22nd USENIX Security Symposium, Washington, DC, Aug. 14–16, 2013*, pp. 589–604.

81. A. Mohaisen, "Towards automatic and lightweight detection and classification of malicious web contents," in *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies, Washington, DC, Nov. 12–15, 2015*, pp. 67–72.

82. A. Zarras, A. Papadogiannakis, R. Gawlik, and T. Holz, "Automated generation of models for fast and precise detection of HTTP-based malware," in *Proc. of the 12th Annual International Conference on Privacy, Security, and Trust, Toronto, ON, Canada, July 23–14, 2014*, pp. 249–256.

83. A. Niakanlahiji, B. T. Chu, and E. Al-Shaer, "PhishMon: A Machine Learning Framework for Detecting Phishing Webpages," in Proc. 2018 IEEE International Conference on Intelligence and Security Informatics (ISI), 2018, pp. 220–225.

84. P. Prakash, M. Kumar, R. R. Kompella, and M. Gupta, "Phishnet: Predictive blacklisting to detect phishing attacks," in *2010 Proceedings IEEE INFOCOM, San Diego, CA, May 6, 2010*, pp. 1–5.

85. G. Wang, J. W. Stokes, C. Herley, and D. Felstead, "Detecting malicious landing pages in malware distribution networks," in *Proc. of the 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Budapest, Hungary, Aug. 8, 2013*, pp. 1–11.

86. L. A. T Nguyen, B. L To, H. K. Nguyen, and M. H Nguyen, "Detecting phishing web sites: A heuristic URL-based approach," in *Proc. of the 2013 International Conference on Advanced Technologies for Communications (ATC 2013), Hi Chi Minh City, Vietnam, Jan. 6, 2013*, pp. 597–602.

87. I. Ghafir, and V. Prenosil, "Blacklist-based malicious ip traffic detection," in *Proc. of the 2015 Global Conference on Communication Technologies, Thuckalay, India, Dec. 3, 2015*, pp. 229–233.

88. P. Seshagiri, A. Vazhayil, and P. Sriram, "AMA: Static code analysis of web page for the detection of malicious scripts," *Procedia Computer Science*, vol. 93, pp.768–773, December 2016.

89. G. Sonowal, and K. S. Kuppusamy. "PhiDMA–A phishing detection model with multi-filter approach," *Journal of King Saud University-Computer and Information Sciences*, vol. 32, no. 1, pp. 99–112, January 2017.

90. A. Nappa, Z. Xu, M. Z. Rafique, J. Caballero, and G. Gu, "CyberProbe: Towards internet-scale active detection of malicious servers," in *Proc. of the 2014 Network and Distributed System Security Symposium, NDSS 2014, San Diego, CA, Feb. 23–26, 2014*, pp. 1-15.

91. S. Marchal, K. Saari, N. Singh, and N. Asokan, "Know your phish: Novel techniques for detecting phishing sites and their targets," in *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS), Nara, Japan, Aug. 11, 2016*, pp. 323–333.

92. P. McCullagh and J. A. Nelder, *Generalized Linear Models*. Boca Raton, FL: Chapman & Hall, 1989.

93. W. S. Noble, "What is a support vector machine?" *Nature Biotechnology*, vol. 24, no.12, pp.1565–1567, December 2006.

94. J. H. Friedman, "Stochastic Gradient Boosting," *Computational Statistics & Data Analysis*, vol. 38, no. 4, pp. 367–378, Feb. 28, 2002.

95. A. Y. Daeef, R. B Ahmad, Y. Yacob, Y, and N. Y. Phing, "Wide scope and fast websites phishing detection using URLs lexical features," in *Proc. of the 2016 3rd International Conference on Electronic Design (ICED), Phuket, Thailand, Aug. 11–12, 2016*, pp. 410-415.

96. K. P. Murphy, "Naïve Bayes classifiers," *University of British Columbia*, Oct. 24, 2006. [Online.] Available: https://www.cs.ubc.ca/~murphyk/Teaching/CS340-Fall06/reading/NB.pdf. [Accessed: Sept. 19, 2019.]

97. P. Geurts, E. Damien, and L.Wehenkel, "Extremely randomized trees," *Machine Learning*, vol. 63, no.1, pp. 3–42, April 2006.

98. L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, October 2001.

99. J. Girones, "J48 decision tree," [Online]. Available: http://data-mining.business-intelligence.uoc.edu/home/j48-decision-tree. [Accessed: Sept. 19. 2019].

100. N. Friedman, D. Geiger, and M. Goldszmidt. "Bayesian network classifiers," *Machine Learning*, vol. 29, no. 2-3, pp. 131–163, November 1997.

101. R. E. Schapire, “A brief introduction to boosting,” in *IJCAI ’99: Proc. of the Sixteenth International Joint Conference on Artificial Intelligence, vol. 2,* San Francisco, CA: Morgan Kaufman Publishers, Inc., 1999, pp. 1401–1406.

102. G. Tsoumakas, and I. Vlahavas, “Random k-labelsets: An ensemble method for multilabel classification,” in *Proc. of the 18th European Conference on Machine Learning, Warsaw, Poland, Sept. 17–21, 2007*, pp. 406–417.

103. M. L. Zhang, and Z. H Zhou, “Ml-knn: A lazy learning approach to multi-label learning,” *Pattern Recognition*, vol. 40, no.7, pp. 2038–2048, July 2007.

104. N. Tóth, and B. Pataki, “Classification confidence weighted majority voting using decision tree classifiers,” *International Journal of Intelligent Computing and Cybernetics*, vol. 1, no. 2, pp. 169–192, June 6, 2008.

105. D. G Denison, B. K. Mallick, and A. F. Smith, “A Bayesian CART algorithm.” *Biometrika*. vol. 85, no 2. pp. 363–377, June 1998.

106. N. S Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, “On large-batch training for deep learning: Generalization gap and sharp minima,” presented at ICLR 2017: International Conference on Learning Representations, Toulon, France, April 24–26, 2017.

107. S. Shalev-Shwartz, “Online learning and online convex optimization,” *Foundations and Trends® in Machine Learning*, vol. 4, no.2, pp. 107–194, 2012.

108. I. Stephen, “Perceptron-based learning algorithms,” *IEEE Transactions on Neural Networks*, vol. 50, no. 2, pp. 179–191, June 1990.

109. K. Crammer, O. Dekel, J. Keshet, S. Shaley-shwartz, and Y. Singer, “Online passive-aggressive algorithms,” *Journal of Machine Learning Research*, vol. 7, pp. 551–585, March 2006.

110. J. R Quinlan, “Bagging, boosting, and C4.5,” in *AAAI ’96: Proceedings of the 13th National Conference on Artificial Intelligence,* vol. 1, pp. 725–730, August 1996.

111. L. E Peterson, “K-nearest neighbor,” *Scholarpedia.org*, vol. 4, no. 2, 2009. [Online.] Available: http://www.scholarpedia.org/article/K-nearest_neighbor. [Accessed: Aug. 1, 2018].

112. Alexa.com, “The top 500 sites on the web.” [Online]. Available: https://www.alexa.com/topsites. [Accessed: August 2018].

113. PhishTank.com. “Join the fight against phishing.” [Online]. Available: https://www.phishtank.com/. [Accessed: April 8, 2019].

114. Y. Pan and X. Ding, “Anomaly based web phishing page detection,” in *Proc. of the 2006 22nd Annual Computer Security Applications Conference (ACSAC ’06), Miami Beach, FL, Dec. 11–15, 2006*, pp. 381–392.

115. S. A Onashoga, A. Abayomi-Alli, O. Idowu, and J. O. Okesola, “A hybrid approach for detecting for detecting malicious web pages using decision tree and naïve Bayes algorithms,” *Georgian Electronic Scientific Journal: Computer Science & Telecommunications*, vol. 48, no.2, January 2016.

116. C. Liu, L. Wang, B. Lang, and Y. Zhou, “Finding effective classifier for malicious URL detection,” in *ICMSS 2018: Proc. of the 2nd International Conference on Management Engineering, Software Engineering and Service Sciences*, New York: Association for Computing Engineering, 2018, pp. 240–244.

117. D. Abraham and N. S, Raj, “Approximate string matching algorithm for phishing detection,” in *Proc. of the 2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI 2014), Noida, India, Sept. 24–17, 2014*, pp. 2285–2290.

118. J. Wang, Y. Xue, Y. Liu, and T. H. Tan, “JSDC: A hybrid approach for Javascript malware detection and classification,” in *ASIA CCS ’15: Proc. of the 10th ACM Symposium on Information, Computer and Communications Security, Singapore, April 2015*, pp. 109–120.

119. A. Fass, R. P. Krawczyk, M. Backes, and B. Stock, “JaSt: Fully Syntactic Detection of Malicious (Obfuscated) JavaScript,” in *DIMVA 2018: Detection of Intrusions and Malware, and Vulnerability Assessment*, C. Giuffrida, S. Bardin, G. Blanc, Eds., Cham, Switzerland: Springer, 2018, pp. 303–325.

120. Y. S. Hwang, J. B. Kwon, J. C. Moon, and S. J. Cho, “Classifying malicious web pages by using an adaptive support vector machine,” *Journal of Information Processing Systems*, vol. 9, no. 3, pp. 395–404, June 2013.

121. MillerSmiles.co.uk, “The web’s dedicated anti-phishing service,” 2019. [Online]. Available: http://www.millersmiles.co.uk/. [Accessed: Sept. 19, 2019].

122. Castlecops.com, 2019. [Online]. Available: http://castlecops.com. Discontinued. [Accessed: Sept. 19, 2019].

123. 3sharp.com. [Online]. Available: http://www.3sharp.com/projects/antiphishing/. [Accessed: Sept. 19, 2019].

124. Dmoz.org, 2017. [Online]. Available: http://dmoz-odp.org/. [Accessed: April. 8, 2019].

125. Yahoo.com. [Online]. Available: https://www.yahoo.com/. [Accessed: Sept. 19, 2019].

126. HauteSecure.com,. [Online]. Available: http://hautesecure.com/ [Accessed: Sept. 19, 2019].

127. D. S. Anderson, C. Fleizach, S. Savage, G. M, Voelker, "Spamscatter. Characterizing internet scam hosting infrastructure," in *SS '07: Proc. of 16th USENIX Security Symposium on USENIX Security Symposium*, Berkeley, CA: USENIX Association, 2007, pp. 1–16.

128. A Malware Dataset, [Online]. Discontinued. Formerly available: mwm.rising.com.cn. [Accessed: Sept. 19, 2019].

129. Knownsec Information Technology Co. [Online]. Discontinued. Formerly available: Knownsec.com. [Accessed: Sept. 19, 2019].

130. MalwareDomainList.com. [Online]. Available: https://www.malwaredomainlist.com. [Accessed: April. 8, 2019].

131. Google, Malicious Emails from Gmail, 2019. [Online]. Available: www.gmail.com. [Accessed: Sept. 19, 2019].

132. Google Safe Browsing. [Online]. Available: https://safebrowsing.google.com/. [Accessed: Sept. 19, 2019].

133. Wepawet Database. [Online]. Discontinued. Formerly available: http://wepawet.cs.ucsb.edu. [Accessed: Sept. 19, 2019].

134. Google.com. [Online]. Available: www.google.com. [Accessed: Sept. 19, 2019].

135. Netcraft.com. [Online]. Available: https://www.netcraft.com/anti-phishing/. [Accessed: Sept. 19, 2019].

136. Telecom Sudparis. [Online]. Available: https://www.telecom-sudparis.eu/en/aris. [Accessed: Sept. 19, 2019].

137. APWG.org. [Online]. Available: https://apwg.org. [Accessed: Sept. 19, 2019].

138. Blade Defender. [Online]. Available: blade-defender.org. [Accessed: Sept. 19, 2019].

139. CleanMX. [Online]. Available: https://support.clean-mx.com/clean-mx/viruses.php. [Accessed: April 8, 2019].

140. Jowein.de. "Spam domain blacklist (filtered by jwSpamSpy)," last updated Aug. 7, 2020. [Online]. Available: http://www.joewein.de/sw/blacklist.htm. [Accessed: April 8, 2019].

141. MalwareDomains.com, “DNS-BH – Malware Domain Blocklist by RiskAnalytics,” June 11, 2018. Available: https://www.malwaredomains.com/. [Accessed: Sept. 19, 2019].

142. MalwareURL.com. [Online]. Available: https://www.malwareurl.com/. [Accessed: April 8, 2019].

143. URL-domain.txt.malware.com. [Online]. Discontinued. Formerly available: url-domain.txt.malware.com. [Accessed: Sept. 19, 2019].

144. ZeusTracker. [Online]. Discontinued. Formerly available: https://zeustracker.abuse.ch/. [Accessed: Sept. 19, 2019].

145. Compuweb. [Online]. Discontinued. Formerly available: http://compuweb.com/. [Accessed: Sept. 19, 2019].

146. SpyEye Tracker. [Online]. Discontinued. Formerly available: https://spyeyetracker.abuse.ch/. [Accessed: Sept. 19, 2019].

147. VirusTotal.com. [Online] Available: https://www.virustotal.com/. [Accessed: April 8, 2019].

148. VxHeavens. [Online]. Discontinued. Formerly available: http://vxheavens.com/. [Accessed: Sept. 19, 2019].

149. OpenMalware.org. [Online]. Available: openmalware.org. [Accessed: Sept. 19, 2019].

150. WebInspector.com. [Online]. Available: http://www.webinspector.com// [Accessed: Sept. 19, 2019].

151. Korea Internet and Security Agency. [Online]. Available: https://www.kisa.or.kr/eng/main.jsp// [Accessed: Sept. 19, 2019].

152. Github.com, “AnubisMalware.” [Online]. Available: https://github.com/fs0c131y/AnubisMalware. [Accessed: Sept. 19, 2019].

153. hpHosts. [Online]. Discontinued. Formerly available: https://www.hosts-file.net/. [Accessed: Sept. 19, 2019].

154. Intel.com. [Online]. Available: https://www.intel.com. [Accessed: Sept. 19, 2019].

155. OpenPhish.com. [Online]. Available: https://openphish.com/. [Accessed: Sept. 19, 2019].

156. M. Maurer, “Phishload: The Phishload phishing test database, 2012. [Online]. Available: https://www.medien.ifi.lmu.de/team/max.maurer/files/phishload/. [Accessed: Sept. 19, 2019].

157. Digg58. [Online]. Discontinued. Formerly available: www.digg58.com. [Accessed: Sept. 19, 2019].

158. Federal Office for Information Security, “Taking advantage of opportunities – avoiding risks.” [Online]. Available: https://www.bsi.bund.de/EN/Home/home_node.html. [Accessed: Sept. 19, 2019].

159. R. B. Basnet, A. H. Sung, and Q. Liu, “Learning to detect phishing URLs,” *International Journal of Research in Engineering and Technology*, vol. 3, no. 6, June 2014, pp. 11–24.

160. D. Huang, K. Xu, and J. Pei, “Malicious URL detection by dynamically mining patterns without pre-defined elements,” *World Wide Web*, vol. 17, no. 6, pp. 1375–1394, November 2014.

161. R. B. Basnet, and T. Doleck, “Towards developing a tool to detect phishing URLs: a machine learning approach,” in *Proc. of the 2015 IEEE International Conference on Computational Intelligence & Communication Technology, Ghaziabad, India, Feb. 13–14, 2015*, pp. 220–223.

162. K. Rieck, G. Schwenk, T. Limmer, T. Holz, and P. Laskov, “Botzilla: Detecting the phoning home of malicious software,” in *Proc. of the 2010 ACM symposium on applied computing (SAC ’10), Sierre, Switzerland, March 22–25, 2010*, pp. 1978–1984.

163. Github.com, “VERMONT – VERsatile MONitoring Tool.” [Online]. Available: https://github.com/tumi8/vermont/. [Accessed: May 20, 2019].

164. M. Balduzzi, M. Egele, E. Kirda, D. Balzarotti, and C. Kruegel, “A solution for the automated detection of clickjacking attacks,” in *ASIACCS ’10: Proc. of the 5th ACM Symposium on Information, Computer and Communications Security, Beijing, China, April 13–16, 2010*, pp. 135–144.

165. B. Eshete, A. Villafiorita, K. Weldemariam, and M. Zulkernine, “EINSPECT: Evolution-guided analysis and detection of malicious web pages,” in *COMPSAC ’13: Proc. of the 2013 IEEE 37th Annual Computer Software and Applications Conference*, Washington DC: IEEE Computer Society, pp. 375–380.

166. C. Wu, L. I. Min, Y. E. Li, X. Zou, and B. QIANG. “Malicious Website Detection Based on URLs Static Features,” in *Proc. of the 2018 International Conference on Modeling, Simulation, and Optimization (MSO 2018), Shenzhen, China, Jan. 21–22, 2018*.

167. F. Douglis, A. Feldmann, B. Krishnamurthy, and J.C. Mogul, “Rate of change and other metrics: A live study of the world wide web,” in *USITS ’97: Proc. of the USENIX Symposium on Internet Technologies and Systems, Monterey, CA, December 1997*, 14 pages.

168. J. Cho, and H. Garcia-Molina, “The evolution of the web and implications for an incremental crawler,” in *VLDB ’00: Proceedings of the 26th International Conference on Very Large Data Bases*, A. Abadi, M. Brodie, Eds., San Francisco: Morgan Kaufmann Publishers Inc., 2000, pp. 200–209.

169. D. Fetterly, M. Manasse, and M. Najork, M., 2003, November. “On the evolution of clusters of near-duplicate web pages,” In *Proceedings of the IEEE/LEOS 3rd International Conference on Numerical Simulation of Semiconductor Optoelectronic Devices*, IEEE, 2003, pp. 37-45.

170. D. Fetterly, M. Manasse, M. Najork, and J. L. Wiener, “A large-scale study of the evolution of web pages," *Software: Practice and Experience*, vol. 34, no. 2, February 2004, pp. 213–237.

171. A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig, “Syntactic clustering of the web,” *Computer Networks and ISDN Systems*,” vol. 29, no. 8–3, September 1997, pp. 1157–1166.

172. B. E. Brewington and G. Cybenko, “How dynamic is the web?” *Computer Networks*, vol. 33, no.1–6, June 2000, pp. 257–276.

173. L. Lim, M. Wang, S. Padmanabhan, J. S. Vitter, and R. Agarwal, “Characterizing web document change,” in *WAIM ’01: Proc. of the Second International Conference on Advances in Web-Age Information Management*, X. S. Wang, G. Yu, H. Lu, Eds. Berlin: Springer, 2001, pp. 133–144.

174. E. Adar, J. Teevan, S. T. Dumais, and J. L. Elsas, “The web changes everything: Understanding the dynamics of web content,” in *WSDM ’09: Proc. of the Second ACM International Conference on Web Search and Data Mining, Barcelona, Spain, February 2009*, pp. 282–291.

175. S. Y. Kwon, S. H. Lee, and S. J. Kim, “A precise metric for measuring how much web pages change,” in *Proc. of the 11th International Conference on Database Systems for Advanced Applications (DASFAA 2006), Singapore, April 12–15, 2006*, pp. 557–571.

176. Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman, “A search engine backed by internet-wide scanning,” in *CCS ’15: Proc. of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, October 2015*, New York: Association for Computing Machinery, pp. 542–553.

177. Cisco.com, “Cisco Talos: Meet Cisco Talos, the industry-leading threat intelligence group fighting the good fight.” [Online]. Available: https://www.cisco.com/c/en/us/products/security/talos.html. [Accessed: May 20, 2019].

178. Statsmodels.org. “Variance inflation factor, VIF, for one exogenous variable.” [Online]. Available:

https://www.statsmodels.org/devel/generated/statsmodels.stats.outliers_influence.variance_inflation_factor.html. [Accessed: April 8, 2019].

179. M. O. Akinwande, H. G. Dikko, and A. Samson, "Variance inflation factor: As a condition for the inclusion of suppressor variables(s) in regression analysis," *Open Journal of Statistics*, vol. 5, no. 7, pp. 754–767, January 2015.

180. A. G. Asuero, A. Sayago, and A. G. Gonzalez, "The correlation coefficient: An overview," *Critical Reviews in Analytical Chemistry*, vol. 36, no. 1, pp. 41–59, January 12, 2007.

181. XGBoost, "XGBoost documentation." [Online].

182. T. G. Dietterich, "Ensemble methods in machine learning," in *MCS 2000: Multiple Classifier Systems*, vol. 1857, Lecture Notes in Computer Science. Berlin: Springer. pp. 1–15.

183. C. M. Bishop, *Neural Networks for Pattern Recognition*. New York: Oxford University Press, 1995.

184. L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, August 1996.

185. C. Lee, "Feature importance measures for tree models – Part 1," Oct. 28, 2017. [Online]. Available https://medium.com/the-artificial-impostor/feature-importance-measures-for-tree-models-part-i-47f187c1a2c3. [Accessed: Sept. 19, 2019].

186. N. V. Chawla, K. W Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, June 1, 2002.

187. G. Lemaitre, F. Nogueira, D. Oliveira, and C. Aridas, "SMOTE - Synthetic minority over-sampling technique." [Online]. Available: https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.over_sampling.SMOTE.html. [Accessed: Feb. 8, 2019.]

188. Featuretools.com. [Online]. Available: https://www.featuretools.com/. [Accessed: Feb. 8, 2019].

189. L. Sullivan, "Correlation and linear regression." [Online]. Available: http://sphweb.bumc.bu.edu/otlt/MPH-Modules/BS/BS704_Correlation-Regression/BS704_Correlation-Regression_print.html. [Accessed: Feb. 8, 2019].

190. J. Friedman, T. Hastie, and R. Tibshirani, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer Series in Statistics. New York: Springer Series in Statistics, 2009.

191. D. Ruta, and B. Gabrys, “Classifier selection for majority voting,” *Information Fusion*, vol.6, no.1, pp. 63–81, March 2005.

192. P. Baldi, “Autoencoders, unsupervised learning, and deep architectures,” in *UTLW ’11: Proc. of the 2011 International Conference on Unsupervised and Transfer Learning*, vol. 27, pp. 37–49, July 2011.

193. Cymon.io, “Open threat intelligence.” [Online]. Available: https://cymon.io/. [Accessed: Jan. 15, 2019].

194. H. Hsu, and P. A Lachenbruch, “Paired t test,” *Encyclopedia of Biostatistics*, Hoboken, NJ: John Wiley & Sons, 2008.

195. DataNovia.com, “T-test essentials, Definition, formula and calculation,” 2018. [Online]. Available: https://www.datanovia.com/en/lessons/t-test-formula/ [Accessed: July 6, 2020].

196. F. J. Massey, Jr., “The Kolmogorov-Smirnov test for goodness of fit,” *Journal of the American Statistical Association*, vol. 46, no. 253, pp. 68–78, 1951.

197. National Institute of Standards and Technology, “Kolmogorov-Smirnov Two-Sample,” Oct. 9, 2015. [Online]. Available: https://www.itl.nist.gov/div898/software/dataplot/refman1/auxillar/ks2samp.htm. [Accessed: July 7, 2020].

198. F. W. Scholz, and M. A Stephens, “K-sample Anderson–Darling tests,” *Journal of the American Statistical Association*, vol. 82, no. 399, pp. 918–924, 1987.

199. National Institute of Standards and Technology, “Anderson-Darling k sample test,” Oct. 9, 2015. [Online]. Available: https://www.itl.nist.gov/div898/software/dataplot/refman1/auxillar/andeksam.htm. [Accessed: July 6, 2020].

200. P. E. McKight and J. Najab, “Kruskal-Wallis test,” in *The Corsini Encyclopedia of Psychology*, vol. 4, C. B. Nemeroff, L. M. Miller, C. B. Nemeroff, W. E. Craighead, Eds., New York: John Wiley & Sons, 2010.

201. S. Glen, “Kruskal Wallis H Test: Definition, examples, & assumptions,” *StatisticsHowTo.com: Statistics for the Rest of Us*, Feb. 4, 2016. [Online]. Available: https://www.statisticshowto.com/kruskal-wallis/ [Accessed: July 6, 2020].

202. M. Arab, and M. K. Sohrabi, “Proposing a new clustering method to detect phishing websites,” *Turkish Journal of Electrical Engineering & Computer Sciences*, vol. 25, no. 6, January 2017, pp. 4757-4767.

203. Dzone.com, “What is JavaScript obfuscation and when is it used?” Jan. 22, 2018. [Online]. Available: https://dzone.com/articles/obfuscation-what-is-obfuscation-in-javascript-why/. [Accessed: May 20, 2019].

204. P. Mioni, “Anatomy of a malicious script: How a website can take over your browser,” July 13, 2018. [Online]. Available: https://css-tricks.com/anatomy-of-a-malicious-script-how-a-website-can-take-over-your-browser/ [Accessed: May 20. 2019].

205. StackOverflow.com, “Malicious JavaScript code,” April 5 2013. [Online]. Available: https://stackoverflow.com/questions/15825408/malicious-javascript-code/. [Accessed: May. 1, 2019].

206. Mozilla MDN Web Docs, “HTTP headers,” April 27, 2020. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers. [Accessed: Aug. 1, 2019].

207. W3Schools.com, “HTML: The language for building web pages.” [Online]. Available: https://www.w3schools.com/. [Accessed: Aug. 1, 2019].

208. Python Software Foundation, “Python bindings for Selenium.” [Online]. Available: https://pypi.org/project/selenium/. [Accessed: Aug. 1, 2018].

209. S. Yadav, A.K.K Reddy, A.L. Narasimha Reddy, and S. Ranjan, “Detecting algorithmically generated malicious domain names,” in *IMC ’10: Proc. of the 10th ACM SIGCOMM Conference on Internet Measurement, Melbourne, Australia, November 2010,* pp. 48–61.

210. Github.com, “English words.” Available: https://github.com/dwyl/english-words/. [Accessed: Aug. 1, 2018].

211. Webopedia, “TLD – Top-level domain,” 2019. [Online] Available: https://www.webopedia.com/TERM/T/TLD.html. [Accessed: May 20, 2019].

212. Z. Xu, A. Nappa, R. Baykov, G. Yang, J. Caballero. and G. Gu, “AUTOPROBE: Towards automatic active malicious server probing using dynamic binary analysis,” in *CCS ’14: Proc. of the 2014 ACM SIGSAC Conference on Computer and Communications Security ACM, Scottsdale, AZ, November 2014*, pp. 179–190.

213. T. Bujlow, M. T. Riaz, and J. M. Pedersen, “A method for classification of network traffic based on C5.0 machine learning algorithm,” in *ICNC ’12: Proc. of the 2012 International Conference on Computing, Networking and Communications, Maui, HI, Jan. 30–Feb. 12, 2012,* pp. 237–241.

214. G. Gu, J. Zhang, and W. Lee. “BotSniffer: Detecting botnet command and control channels in network traffic,” in NDSS 2008: Proc. of the 15th Annual Network and Distributed System Security Symposium, San Diego, CA, Feb. 10–15, 2008.

215. B. Soniya and M. Wilscy, “Detection of randomized bot command and control traffic on an end-point host,” *Alexandria Engineering Journal,* vol. 55, no. 3, pp. 2771–2781, September 2016.

216. K. Reitz, I. Cordasco, N. Prewitt, “Requests: HTTP for humans,” 2019. [Online]. Available: https://requests.readthedocs.io/en/master/. [Accessed: Aug. 10, 2018].

217. L. Bottou and Y. L. Cun, “Large scale online learning,” in *Advances in Neural Information Processing Systems 16 (NIPS 2003),* S. Thrun, L. K. Saul, and B. Scholkopf, Eds. San Diego, CA: Neural Information Processing Systems, 2004, pp. 217–224.

218. T. Li, G. Kou, and Y. Peng, “Improving malicious URLs detection via feature engineering: Linear and nonlinear space transformation methods,” in *Information Systems,* 91, p.101494, 2020.

219. E. B Beigi, H. H. Jazi, N. Stakhanova, and A. A Ghorbani,“Towards effective feature selection in machine learning-based botnet detection approaches,” in *Proc. of the 2014 IEEE Conference on Communications and Network Security, San Francisco, CA, Oct. 29–31, 2014*, pp. 247–255.

220. The Honeypot Project, “Capture HPC.” [Online]. Available: https://www.honeynet.org/projects/old/capture-hpc/. [Accessed: Mar. 31, 2020].

221. G. Willard, “Understanding the co-evolution of cyber defenses and attacks to achieve enhanced cybersecurity,” *Journal of Information Warfare,* vol. 14., no. 2, pp. 17–31, April 2015.

222. W3Schools.com, “HTML tutorial.” [Online]. Available: https://www.w3schools.com/html/html5_intro.asp. [Accessed: March 31, 2020].

223. K. Pearson, “On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling,” in *Breakthroughs in Statistics,* S. Kotz and N.L. Johnson, Eds., New York: Springer, 1992, pp. 11–28.

224. I. Syarif, A. Prugel-Bennett, and G. Wills, “SVM parameter optimization using grid search and genetic algorithm to improve classification performance,” *Telkomnika,* vol. *14,no.* 4, p.1502, 2016.

225. M. Butkiewicz, H. V. Madhyastha, and V. Sekar, “Understanding website complexity: measurements, metrics, and implications,” in *ICM ’11: Proc. of the 2011 ACM SIGCOMM Conference on Internet Measurement, Berlin, Germany, Nov. 2–4, 2011*, pp. 313–328.

226. D. K. McGrath, and M. Gupta, “Behind phishing: An examination of phisher modi operandi,” in *Proc. of the First USENIX Workshop on Large-Scale Exploits and Emergent Threats, San Francisco, CA, April 15, 2008*, p. 4.

227. University of Notre Dame, “The frequency of the letters of the alphabet in English.” [Online]. Available: https://www3.nd.edu/~busiforc/handouts/cryptography/letterfrequencies.html. [Accessed: Mar. 31, 2020].

228. Noxxi, “HTTP evasions explained.” [Online]. Available: https://noxxi.de/research/semantic-gap.html. [Accessed: Mar. 31, 2020].

229. OWASP.org, “Cache poisoning.” [Online]. Available: https://owasp.org/www-community/attacks/Cache_Poisoning. [Accessed: Mar. 31, 2020].

230. “Malicious Apache server and Blackhold provide stealthy backdoor,” *Info Security*, April 30, 2013. [Online]. Available: https://www.infosecurity-magazine.com/news/malicious-apache-server-and-blackhole-provide/ [Accessed: Mar. 31, 2020].

231. C. Cimpanu, “Apache web server bug grants root access on shared hosting environments,” *Zero Day*, April 3, 2019. [Online]. Available: https://www.zdnet.com/article/apache-web-server-bug-grants-root-access-on-shared-hosting-environments/. [Accessed: Mar. 31, 2020].

232. D. Goodin, “Rampant Apache website attack hits visitors with highly malicious software,” *Ars Technica*, July 3, 2013. [Online]. Available: https://arstechnica.com/information-technology/2013/07/darkleech-infects-40k-apache-site-addresses/. [Accessed: Mar. 31, 2020].

233. L. Tung, “’Sophisticated’ backdoor malware opens up security blackhole in Apache webservers,” *ZdNet.com*, May 1, 2013. [Online]. Available: https://www.zdnet.com/article/sophisticated-backdoor-malware-opens-up-security-blackhole-in-apache-web-servers. [Accessed: Mar. 31, 2020].

234. ThreatX Labs, “Malicious bot detection through a complex proxy network,” *Security Boulevard*, April 17, 2019. [Online]. Available: https://securityboulevard.com/2019/04/malicious-bot-detection-through-a-complex-proxy-network/. [Accessed: Mar. 31, 2020].

235. E. J. Zaborowski, “Malicious proxies: The web’s evil twin,” *Defcon.org*, 2009. [Online]. Available: https://www.defcon.org/images/defcon-17/dc-17-presentations/defcon-17-edward_zaborowski-doppelganger.pdf. [Accessed: Mar. 31, 2020].

236. H. Pandjarov, “How the vary HTTP header can be bad,” *SiteGround*, June 21, 2017. [Online]. Available: https://www.siteground.com/blog/vary-http-header/. [Accessed: Mar. 31, 2020].

237. E. Kovaks, “Phishers use new method to bypass Office 365 safe links,” *Security Week*, May 8, 2018. [Online]. Available: https://www.securityweek.com/phishers-use-new-method-bypass-office-365-safe-links. [Accessed: Mar. 31, 2020].

238. Y. Nathaniel, "MetaMorph HTML obfuscation phishing attack," *Avanan*, Aug. 14, 2019. [Online]. Available: https://www.avanan.com/blog/metamorph-html-obfuscation-phishing-attack. [Accessed: Mar. 31, 2020].

239. M. Gualtieri, "Stealing data with CSS: Attack and defense," Feb. 6, 2018. [Online]. Available: https://www.mike-gualtieri.com/posts/stealing-data-with-css-attack-and-defense/. [Accessed: Mar. 31, 2020].

240. "Ponemon Institute reveals security teams spend approximately 25 percent of their time chasing false positives; response times," *Bloomberg*, Aug. 1, 2019. [Online]. Available: https://www.bloomberg.com/press-releases/2019-08-01/ponemon-institute-reveals-security-teams-spend-approximately-25-percent-of-their-time-chasing-false-positives-response-times. [Accessed: Mar. 31, 2020].

241. L. V. D Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, November 2008.

242. K. Pearson, "On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling," in *Breakthroughs in Statistics*, S. Kotz and N.L. Johnson, Eds., New York: Springer, 1992, pp. 11–28.

243. A. C. Acock, and G. R. Stavig, "A measure of association for nonparametric statistics," *Social Forces*, vol. 57, no. 4, pp. 1381–1386, June 1979.

244. H. He, Y. Bai, E. A. Garcia, and S. Li, "ADASYN: Adaptive synthetic sampling approach for imbalanced learning," in *Proc. of the 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), Hong Kong, China, June 1–8, 2008*, pp. 1322–1328.

245. "Esprima 4.0.1: ECMAScript parsing infrastructure for multipurpose analysis in Python," *Python Package Index*. [Online]. Available: https://pypi.org/project/esprima/. [Accessed: Mar. 1, 2020].

246. W. Hoeffding and H. Robbins, "The central limit theorem for dependent random variables," *Duke Mathematical Journal*, vol.15, no. 3, pp.773-780, 1948.

247. P. Lumley, P. Diehr,, S. Emerson, and L. Chen, "The importance of the normality assumption in large public health data sets," *Annual review of public health*, vol. 23, no. 1, pp. 151-169, 2002.

248. E. R. Girden, ANOVA: Repeated Measures, Newbury Park, CA: SAGE Publications, Inc., 1992.