2019 15th European Dependable Computing Conference (EDCC)

A Comprehensive Evaluation of HTTP Header Features for Detecting Malicious Websites

John McGahagan IV
Center for Risk and Reliability
Universty of Maryland
College Park, MD, USA
jmcgahag@umd.edu

Darshan Bhansali
Center for Risk and Reliability
University of Maryland
College Park, MD, USA
darshanb@umd.edu

Margaret Gratian
Center for Risk and Reliability
Universty of Maryland
College Park, MD, USA
mgratian@umd.edu

Michel Cukier
Center for Risk and Reliability
Universty of Maryland
College Park, MD, USA
mcukier@umd.edu

Abstract— **Security researchers have used website features including the URL, webpage content, HTTP headers, and others to detect malicious websites. In prior research, features derived from HTTP headers have shown promise for malicious website detection. This paper includes a comprehensive evaluation of HTTP header features to assess whether additional HTTP header features improve malicious website detection. We analyze HTTP headers from 6,021 malicious and 39,853 benign websites. We define malicious websites as those identified by Cisco Talos Threat Intelligence Group for association with phishing, drive-by downloads, and command and control infrastructure. Benign websites consist of popular websites from the Alexa Traffic Rank. We collect 672 HTTP header features from these websites and identify 22 for further analysis. Among these, 11 have been studied in prior research while the other 11 are new and identified in our research. From these 22 features, eight features, three identified by our study, consistently rank as the most important features and represent 80% of the total feature importance. We build eight models with supervised learning techniques and observe that the detection performance metrics for the 22 features are consistently better than for the 11 previously studied features. We also apply two feature transformation techniques and find that performing Principal Component Analysis on the features identified increases detection ability. From our results, we postulate that use of additional HTTP header features will lead to more accurate detection of malicious websites.**

Keywords—Malicious website, HTTP header, Feature analysis, Web Security, Supervised Learning

I. Introduction

Using website features to detect malicious websites is an active area of research and has potential to identify malicious websites without the use of specific signatures. Website features are diverse and include the website URL, webpage content, and others. Prior research has explored the use of HTTP header features in combination with other features to detect malicious websites. For example, [1] are able to detect malicious websites with a 92.2% accuracy and a false positive rate of 0.1% by using HTTP and domain features. They also evaluated which features are the most important with respect to information gain values and found the number of `requests` for specific `Content-Type`s to be prevalent in their top 10. Authors in [2] also detect malicious websites with several types of website features, including HTTP headers, and achieve a max accuracy of over 96%. They identified the `Server` and `Cache-Control` headers as useful application layer features in their study. HTTP header features have been used for malicious website detection but, to our knowledge, no study has conducted an analysis of all HTTP headers from a website to evaluate the potential of using these features alone to detect malicious websites. We explore this approach for two reasons. First, prior research places little emphasis on finding new HTTP header features for malicious website detection leading to potential missed detection opportunities. For example, `chunked` encodings have been associated with malicious traffic but are absent from prior research. Also, malicious website detection is a major challenge and new methods are needed to defeat evolving threats. Second, HTTP headers can be gathered with minimal overhead from web traffic making them easy to incorporate into signatures created by a security operations center or incident response team.

This paper includes a comprehensive evaluation of an HTTP header-only approach to malicious website detection to assess whether additional HTTP header features can improve malicious website detection. We analyze HTTP headers from 6,021 malicious and 39,853 benign websites. Malicious websites were identified by Cisco Talos Threat Intelligence Group and include websites associated with phishing campaigns, drive-by downloads, and command and control (C2) infrastructure. Benign websites consist of the most popular websites from the Alexa Traffic Rank [3]. We collect 672 HTTP header features from these websites and identify 22 for further analysis; 11 of these have been studied in prior research and the other 11 are newly identified by our approach. We apply eight statistical techniques and account for our dataset imbalance by using no sampling, over-sampling and under-sampling scenarios. We apply two feature transformations to determine if there are combinations of HTTP header features that can help detect malicious websites.

Our contributions are the following:

- We demonstrate the potential of using only HTTP header features as a means to detect malicious websites;
- We introduce 11 new HTTP header features not previously considered to aid in the detection of malicious websites;
- Among the 22 features, eight features, three newly identified by our approach, rank as the most important features and represent 80% of feature importance;

- The average Matthews Correlation Coefficient (MCC) for the selected 22 features is better than the average MCC for the 11 previously studied features across our three sampling scenarios;
- We find that applying Principal Component Analysis (PCA) to the 22 selected features improves malicious website detection.

The remainder of this paper is organized as follows. Section II presents the related work. Section III describes HTTP headers and feature collection. Section IV details our feature selection process. Section V discusses our analysis and results. Section VI provides limitations, and Section VII concludes the paper.

II. Related Work

There are a few common approaches to malicious website detection. Some authors apply simple heuristics, such as in [5]-[7], but this approach is not prevalent. Supervised learning approaches with various algorithms are more common for malicious website detection and have been demonstrated in [1]-[2] and [8]-[20]. In our work we use supervised learning but expand our approach to use eight different algorithms.

Features gathered over a session have been used to identify malicious websites and traffic. Authors in [1] 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 in [2] and [8]-[9] took similar approaches and combined non-HTTP features with specific metadata gathered from the interaction with a website. These approaches obtain detection rates of up to 96%. Reference [21] used the `Content-Type` header as a means of distinguishing between different types of HTTP traffic and [22] 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 approaches limit themselves to a small list of features or require additional non-HTTP features to achieve their performance metrics. Reference [20] considered all HTTP headers as potential features, but they used lengths of the respective headers and we are using header content.

Other approaches demonstrate that HTTP traffic generated by malware can be used to build signatures or fingerprints for detection. Authors in [23]-[24] clustered the HTTP communications generated to and from HTTP-based malware on their testbed to create signatures. Reference [25] recorded HTTP traffic over a session and produced a list of the influential features for malicious traffic identification which consisted of transmission control protocol (TCP) and HTTP features. These authors found the `Content-Length` header to be of importance. ARROW, by [26], generates signatures from redirect chains captured in HTTP traces. Reference [27] took a clustering approach to grouping URLs in the malware-generated HTTP traffic in order to classify the C2 communications. BotHound, by [28], found malicious communications may have similar `User-Agent` strings in `request`s. Generating signatures or fingerprints for malicious HTTP communications is also used in [29]-[33].

While prior approaches have promise, there are inherent limits to these approaches. First, prior research typically only considers a subset of the potential HTTP headers. This could miss other header features that are relevant. Second, when HTTP headers are included, they are often part of a larger feature set. Methods that require multiple types of features – traffic statistics, webpage content, etc, require additional instrumentation and overhead. Lastly, approaches that collect features over a period of time or session have an inherent time constraint. Our approach attempts to address these challenges.

Prior research typically makes use of two paradigms for dataset selection – leverage a well-known dataset of benign or malicious websites or create a custom dataset. References [5], [28], and [34] customize their datasets and [1]-[2], [6], [18], and [35] leverage well-known datasets. There is no standard dataset for evaluation and both paradigms have merit. After considering prior approaches and potential sources including [3] and [36]-[43] we chose to use popular Alexa websites [3] provided by [4] which are prevalent in prior research as our benign dataset and malicious websites from Cisco Talos Intelligence Group as our malicious dataset. Choosing datasets from third parties minimizes unintentional bias that could be introduced when creating or customizing a dataset.

III. HTTP Feature Collection

HTTP headers provide information about a website and about the interactions between a website and a user. An HTTP header is a key-value pair within an HTTP `request` or `response`; `request`s and `response`s may contain multiple headers. The example below shows the HTTP `request` headers generated during a web `request`. **Bold** items are the header names (keys) and non-bold items are the corresponding values. '…' indicates where a value was 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
```

For commonly used headers, please refer to the Mozilla Developer Network [44]. HTTP `response`s have a similar structure of key-value pairs. In this paper, we perform a `GET request` to the selected websites and record the headers present in the `response`. We only consider `response` headers.

HTTP feature collection took place in August 2018. We used the `Python` [45] `requests` library [46] to make `GET requests` to the websites and collected HTTP features in the associated `response`. Upon receiving `response`s, 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 examined the HTTP specification and determined if the header has 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 Boolean value features that capture whether the specific directive is present in the header. Another group of features we gathered is 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 indicates the `response` may be held in any cache and the `max-age` directive is set to 31,536,000 seconds. Our method captures both of these features. Overall, data collection resulted in a total of 672 HTTP features.

IV. Feature Selection

After collection of 672 HTTP header features, we analyzed which features have strong association with the dependent variable (i.e., whether the website is malicious) and eliminated any redundant features (i.e., have no/weak relationship with the dependent variable). First, we removed features for which all the websites' HTTP `response` headers have the same value. There are 399 such features. Next, we attempted to remove features specific to our dataset by removing features that have the same value at least 95% of the time. This eliminated 245 features. We then evaluated the remaining 28 features to identify features which have a high multicollinearity. Removing features with high multicollinearity ensures we analyze a set of independent features. Collinearity can be quantified by the Variance Inflation Factor (VIF) [47]. First, we determined the VIF values for each feature. We then iteratively identified features that have a VIF greater than 5 as used in [48]. Among our list of features with a VIF greater than 5, we then determined which features have similar VIF values and high correlation to each other. We defined high correlation as having a correlation of greater than 0.7 as in [49]. Among the highly correlated features with similar VIF values, we then removed the feature with the highest VIF from our feature set. We are left with our final feature set of 22.

Next, we performed statistical tests to identify the features that have the strongest association with the dependent variable. For the numerical features, we computed the Point Biserial Correlation [50] and performed a t-test of independence. These tests suggested that of the three numerical variables, `options nosniff`, had the strongest association with the dependent variable. We performed Pearson's chi-square test of independence on categorical features. These tests assisted in determining whether we should eliminate a feature and identified which features may be more significant than others. For categorical variables, the tests indicated that `content-encoding gzip` and `content-type text/html` had chi-square values twice that of other variables indicating a strong association with the dependent variable.

V. Feature Analysis

A. Analysis Approach

We created two features sets. The first consists of the total 22 features identified by our approach and the second consists of 11 features identified in our approach that have also been studied in prior research. We evaluated the feature sets against eight different supervised classifiers [51] and recorded their performance metrics. The supervised classifiers belong to several classes of algorithms: Nearest Neighbors, Generalized Linear Models, Ensemble Methods, and Neural Networks (NN)[52]. Among the eight models, four are Ensemble Methods and provide a measure of feature importance: Adaptive (Ada) Boosting (AB) [53], Extra Trees (ET) [54], Random Forest (RF) [55], and Gradient Boosting (GB) [56]. The other models do not provide a measure of feature importance but cover other classes of algorithms. Bagging Classifier (BC) [57] is an Ensemble Method. Logistic Regression (LR) [58] is a Generalized Linear Model. K-Nearest Neighbors (KNN) [59] is a Nearest Neighbor Method.

For all models, we split training and testing data using an 80:20 split which is a common train/test split. Our dataset is imbalanced: we have 39,835 benign and 6,021 malicious websites. To address the imbalance and ensure results are 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 resulted in three different training datasets that we used to evaluate models.

For no sampling, we used 31,853 benign 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 such that we had the same number of malicious as benign websites. For over-sampling, we over-sampled the malicious websites using the SMOTE [60] technique available in the `imbalanced` [61] `Python` library. This method creates new instances of the malicious class like the existing ones using the K-Nearest Neighbors algorithm. We tried various over-sampling ratios for the minority class (0.3, 0.455, 0.606, 0.758, 1.0) in training and present results for the case where the maximum number of malicious websites are identified in the test set. The number of new malicious class instances is equal to the difference between the number of non-malicious instances and existing number of malicious instances in the training set. We get a balanced training set of 31,853 benign and 31,853 malicious websites. The websites used in the testing set remained consistent across all models and sampling approaches so we can compare results. There is no overlap between training and testing datasets.

B. Feature Ranking Analysis

We used Ensemble Methods (RandomForest, AdaBoost, ExtraTree, and Gradient Boosting) to understand feature importance, a normalized metric between 0 and 1.0 that can provide a ranking of how useful features are to a classifier for differentiating between malicious and benign websites. Table I presents the 22 features with their average rank and importance in the sampling scenarios using the four Ensemble Methods. New features are shaded and underlined. We observed that the top two features, which are prevalent in prior research, have an importance much higher than the remaining 20 features (`content-length` has an importance of 0.33 and `content-encoding gzip` has an importance of 0.21 without sampling) both with and without sampling. We also observed that the feature rank and importance are similar when considering over-sampling or under-sampling. We also observed that the top eight features are the same with and without sampling. The cumulative importance of these eight features is 0.83, 0.80 and 0.81 for no sampling, over-sampling and under-sampling, respectively.

Of the 22 selected features, 11 were studied in prior research and the remaining 11 are identified by our approach. `HSTS` is an abbreviation for the `strict-transport-security` header. The top two features, which have a much higher importance than all other features, have been studied in prior research. However, the third feature, `transfer-encoding chunked` was identified by our approach. Note that this feature ranks third without sampling and when under-sampling, and fourth when over-sampling. Similarly, `vary accept` ranked fifth, third, and fourth when not sampling, over-sampling, and under-sampling, respectively. Of the five top features, two are newly identified in our study. Such observations hint to possible detection improvement when including the new features.

Five of the new features have security implications. `chunked` data is considered a way to evade security scanners. This feature has not been studied in prior research but is ranked highly. The `expect-ct` header prevents mis-issued certificates from going unnoticed [44]. `HSTS` informs the browser that the website should only be accessed over HTTPS. The presence of `x-xss-protection` tells the browser to stop loading the webpage if a cross-site scripting attack is detected. The `x-content-type` header with value of `nosniff` can prevent MIME 'sniffing' attacks. To our knowledge, the other six features do not have specific security motivation but their appearance on our list motivates further evaluation.

TABLE I. ALL FEATURES RANK AND IMPORTANCE

All Features Ranked in Sampling Scenarios			
Feature	***No***	***Over***	***Under***
`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)
`HSTS 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 II provides the feature rank and importance for the 11 features from prior research. Compared to the 22 features, the first two features have a 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 ranges between 0.66 and 0.72 for no sampling, over-sampling, and under-sampling. As for the 22 features, the feature rank and importance are similar when considering over-sampling or under-sampling. We also observe that the top five features are the same with and without sampling. The overall importance of these five features is 0.91, 0.88, and 0.91 for no, over and under-sampling, respectively.

TABLE II. PRIOR FEATURES RANK AND IMPORTANCE

Prior Features Ranked in Sampling Scenarios			
Feature	***No***	***Over***	***Under***
`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)

C. *Model Performance Comparison*

We investigated model performance for the test dataset across our three sampling scenarios. All models were built with default parameters provided by [62]. We compared the results of using the 22 features to the results of using the 11 previously studied features. Tables III and IV provide the false positive rate (FPR), false negative rate (FNR), accuracy (ACC), area under the receiver operating characteristic curve (AUC), and MCC. AUC plots the true positive rate vs false negative rate. MCC measures the quality of a binary classifier and ranges between -1 and 1, with 1 representing a perfect classifier, 0 a random classifier, and -1 indicating complete disagreement between the predicted and actual value. MCC drives our discussion because it is a function of the true positive/negative rates and the false positive/negative rates, considers all four quadrants of the confusion matrix, and accommodates our imbalanced dataset.

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 22 features instead of the previous studied 11 features. With over-sampling, the MCC was higher for all eight models when considering the 22 features instead of the 11 previously studied features. When under-sampling, the average MCC increased from 0.545 to 0.61 when considering 22 features instead of the previous studied 11 features. With under-sampling, the average MCC was higher for all eight models except Bagging Classifier when considering the 22 features instead of the 11 previously studied features. In each of our sampling scenarios we observed overall improvement when incorporating the 11 newly identified features. Over-sampling slightly improves the average MCC (0.62 instead of 0.615) and under-sampling slightly decreased it (0.61) when considering 22 features.

D. *Feature Transformation*

We performed feature transformation on the 22 features to determine if there are combinations of features that improve performance. We have 19 categorical features; thus, log transformations would lead to many undefined values. The data for our numerical features also led to many undefined values. Hence, we used only basic pair-wise feature transformations: addition, multiplication, and division, with the `Python` library, `featuretools` [63]. The original 22 features were transformed into 946 features. We then performed feature elimination on the 946 features using four different techniques: Correlation [64], SelectKBest (scoring function chi-square), RecursiveFeatureEliminaton, and SelectFromModel [62]. We used features selected by at least 3 of these techniques, leaving 36 in total. We then determined if PCA could reduce the 946 features to 'n' components and capture the maximum variance. Using a cumulative scree plot, we identified 117 components that capture 95% of the variance and proceeded to build our eight models with these 117 components.

Table IV shows the results of the feature transformations. Feature transformation with feature selection increased the MCC for all models, except AdaBoost, when considering the 22 features instead of the 11 previously studied features and the average MCC improved from 0.53 to 0.56. However, when considering the 22 features, feature transformation with feature selection reduced the average MCC from 0.615 to 0.56 when compared to the no sampling scenario. For PCA, the MCC was higher for the all eight models when considering the 22 features instead of 11 previously studied features and the average MCC improved from 0.59 to 0.65. When compared to the no sampling case, the average MCC improved from 0.615 to 0.65.

When considering the effect of feature transformation on our model performance, we found that feature transformation with feature selection reduced the average MCC and PCA improved the average MCC. Thus, compared to the case without sampling and without feature transformation, PCA improved the results, but feature transformation with feature selection worsened them.

E. *Cross-Validation and Tuning Models*

Lastly, we performed hyperparameter tuning and cross-validation with the goal of validating Tables III and IV. Models in Tables III and IV were built with an 80:20 train/test split. We chose the best performing model in each case from Tables III and IV and proceeded with parameter tuning and cross-validation. We used a Decision Tree Classifier as the base estimator and StratifiedKFold [65] for 10-fold cross-validation [66]. All five models improved but the average MCC only increased from 0.652 to 0.657, suggesting validity of the results in Tables III and IV.

We also sought to add assurance that results were not dependent on the 80:20 split. We did this by repeating the approach in Section V.C followed by parameter tuning and cross-validation of the best models on a 70:30 split of train/test data. 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 with the 70:30 and 80:20 split respectively. With tuning and cross-validation the average MCC was 0.655 and 0.657 with the 70:30 and 80:20 split respectively. The small difference between results in the different splits suggests we are not dependent on the train/test split. Results are shown in Table V.

VI. LIMITATIONS

The first limitation arises from the nature of our dataset. In the related work, authors use different techniques for curating datasets of benign and malicious websites. Some authors implement web crawlers or similar methods to identify and collect websites while others use established datasets. There is inherent subjectivity associated with both approaches for defining and curating datasets of 'benign' and 'malicious' websites. We chose to work with established datasets, using the top 39,835 websites pulled from the most popular global websites based on the Alexa Traffic Rank as our benign websites and 6,021 websites provided to us by the Cisco Talos Threat Intelligence Group as our malicious websites.

By using websites from the Alexa Traffic Rank to define our benign websites, we make the assumption that popularity is equivalent to being benign. While this may not always be the case, we assume that websites that have been created for malicious intent are unlikely to become popular. To verify this assumption, we searched for our list of 39,835 benign websites in Cymon.io [67], a tool that accumulates threat intelligence feeds. In 2018, 2,099 of the 39,835 websites in our benign list appeared in Cymon.io's database. Appearance in Cymon.io's database does not confirm that the website is malicious; similarly, the lack of a website's presence in the database does not confirm the website is benign. However, the fact that only 5.6% of our benign websites are present suggests that using popularity as a proxy for ground truth of benign websites is a reasonable assumption. Lastly, our dataset represents only a snapshot in time. The security space and the internet are dynamic environments and observations made at one time may not be applicable later. This limitation is difficult to avoid when studying websites.

By using websites from the Cisco Talos Threat Intelligence Group data to define our malicious websites, we make the assumption that being malicious implies being associated with phishing, drive-by downloads, or C2 infrastructure. 'Malicious' has no formal definition in cybersecurity, so other researchers or tools may define malicious websites differently. Therefore, there may be different types of malicious websites that our approach does not capture. Furthermore, given that this is a proprietary dataset, we have no knowledge into the breakdown of the types of threats in the dataset. This is a challenge because certain threats have more distinct indicators.

Another limitation is the selection of HTTP header features. 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 that arise from HTTP `request`s and `response`s over a period of time. Also, we did not use HTTP features in combination with other website features. In future work we will expand to other types of website features.

The final limitation is our inability to benchmark the success of our approach against related research. Previous authors used different datasets and methodologies to analyze and classify malicious websites. Thus, direct comparison of results to other work was infeasible. This points to a broader problem in the field of cybersecurity: the lack of repeatability which hinders validation and comparison.

VII. Conclusions and Future Work

This paper includes a comprehensive evaluation of HTTP header features to assess whether additional HTTP header features improve malicious website detection. We analyzed HTTP headers from 6,021 malicious and 39,853 benign websites. Malicious websites were identified by the Cisco Talos Threat Intelligence Group and benign websites included the most popular websites from the Alexa Traffic Rank. We collected 672 HTTP header features from these websites and identified 22 for further analysis, of which, 11 are new. We built eight models and added robustness to our methodology by performing no sampling, over-sampling and under-sampling.

Among the 22 features, eight features, three identified by our approach, consistently ranked as the most important features and represented 80% of feature importance. The average MCC for the selected 22 features was better than for the 11 previously studied features. When considering the 22 selected features, PCA increased the average MCC. Our results indicate that there is a broader set of HTTP header features that can be used for malicious website detection than those commonly studied. Future work includes the analysis of additional features to detect malicious websites.

Acknowledgment

We thank Cisco Talos, TalosIntelligence.com for having created and shared the list of 6,021 malicious websites. We also acknowledge the support of the U.S. Department of Defense. The views and conclusions expressed in this paper are those of the authors and do not necessarily represent those of the Department of Defense or U.S. Government.

References

[1] W. Tao, Y. Shunzheng, and X. Bailin, “A novel framework for learning to detect malicious web pages,” In Proc. 2010 International Forum on Information Technology and Applications, vol. 2. 2010, pp. 353-357.

[2] L. Xu, Z. Zhan, S. Xu, and K. Ye, “Cross-layer detection of malicious websites,” In Proc. Third ACM conference on Data and Application Security and Privacy, 2013, pp. 141–152.

[3] Amazon.com, “The top 500 sites on the web,” [Online]. Available: https://www.alexa.com/topsite/. [Accessed August. 2018].

[4] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman, “A Search Engine Backed by Internet-Wide Scanning,” In Proc. 22nd ACM Conference on Computer and Communications Security, 2015, pp. 542-553.

[5] C. Seifert, I. Welch, and P. Komisarczuk, “Identification of malicious web pages with static heuristics,” In Proc. Telecommunication Networks and Applications Conference, 2008, pp. 91–96.

[6] S. Gastellier-Prevost, G.G. Granadillo, and M. Laurent, “Decisive heuristics to differentiate legitimate from phishing sites,” In Proc. 2011 Conference on Network and Information Systems Security, 2011, pp. 1-9.

[7] 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. 2013 International Conference on Advanced Technologies for Communications, 2013, pp. 597-602.

[8] A. E. Kosba, A. Mohaisen, A. West, T. Tonn, and H.K. Kim, “ADAM: automated detection and attribution of malicious webpages,” In Proc. International Workshop on Information Security Applications, 2014, pp. 3-16.

[9] A. Mohaisen, “Towards automatic and lightweight detection and classification of malicious web contents,” 2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies, 2015, pp. 67-72.

[10] J. Ma, L.K. Saul, S. Savage. and G.M. Voelker, “Beyond blacklists: learning to detect malicious web sites from suspicious URLs,” In Proc. 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2009, pp. 1245-1254.

[11] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, “Identifying suspicious urls: An application of large-scale online learning,” In Proc. International Conference on Machine Learning, 2009, pp. 681–688.

[12] C. Whittaker, B. Ryner, and M. Nazif, “Large-scale automatic classification of phishing pages,” In Proc. of NDSS ’10, 2010.

[13] D. Canali, M. Cova, G. Vigna, and C. Kruegel, “Prophiler: a fast filter for the large-scale detection of malicious web pages,” In Proc. 20th International Conference on World wide web, 2010, pp. 197-206.

[14] X. Guang, O. Jason, P. R. Carolyn, and C. Lorrie, “CANTINA+: A feature-rich machine learning framework for detecting phishing web sites,” In Proc. ACM Transactions on Information and System Security, 2011, pp. 1–28.

[15] X. Gu, W. Hongyuan, and N.I. Tongguang, “An efficient approach to detecting phishing web,” *Journal of Computational Information Systems*, vol. 9 no. 14, Jul., pp. 5553–5560, 2013.

[16] A. Le, A. Markopoulou, and M. Faloutsos, “Phishdef: Url names say it all,” In Proc. 2011 IEEE INFOCOM, 2011, pp. 191-195.

[17] R.B. Basnet, A.H. Sung, and Q. Liu, “Feature selection for improved phishing detection,” In Proc. International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems, 2012, pp. 252-261.

[18] B. Eshete, A. Villafiorita, and K. Weldemariam, “Binspect: Holistic analysis and detection of malicious web pages,” In Proc. International Conference on Security and Privacy in Communication Systems, 2012, pp. 149-166.

[19] 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, Jan., pp. 243-261, 2016.

[20] 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, 2018, pp. 220-225.

[21] T. Bujlow, M.T. Riaz, and J.M. Pedersen, “A method for classification of network traffic based on C5.0 machine learning algorithm,” In Proc. International Conference on Computing, Networking and Communications, 2012, pp. 237–241.

[22] K. Goseva-Popstojanova, G. Anastasovski, A. Dimitrijevikj, R. Pantev, and B. Miller, “Characterization and classification of malicious Web traffic,” *Computers and Security*, vol. 42, May., pp. 92-115, 2014.

[23] R. Perdisci, W. Lee, and N. Feamster, “Behavioral clustering of http-based malware and signature generation using malicious network traces,” In Proc. 7th USENIX conference on Networked Systems Design and Implementation, NSDI’10, 2010, pp. 26.

[24] R. Perdisci, A. Davide, and G. Giacinto, “Scalable fine-grained behavioral clustering of http-based malware,” *Computer Networks*, vol. 57, no. 2, Feb., pp 487-500, 2013.

[25] 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, Dec., 2013.

[26] J. Zhang, C. Seifert, J. W. Stokes, and W. Lee, “Arrow: Generating signatures to detect drive-by downloads,” In Proc. 20th International Conference on World wide web, 2011, pp. 187–196.

[27] N. Kheir, G. Blanc, H. Debar, J. Garcia-Alfaro, and D. Yang, “Automated classification of C&C connections through malware URL clustering,” In Proc. ICT Systems Security and Privacy Protection, 2015, pp. 252–266.

[28] A. Zarras, A. Papadogiannakis, R. Gawlik, and T. Holz, “Automated generation of models for fast and precise detection of HTTP-based malware,” In Proc. 12th Annual International Conference on Privacy, Security and Trust, 2014, pp. 249–256.

[29] G. Gu, J. Zhang, and W. Lee, “BotSniffer:Detecting botnet command and control channels in network traffic,” In Proc. 15th Annual Network and Distributed System Security Symposium (NDSS’08), 2008.

[30] 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, 2010, pp. 31-39.

[31] T. Nelms, R. Perdisci, and M. Ahamad, “Execscent: Mining for new c&c domains in live networks with adaptive control protocol templates,” In Proc. 22nd USENIX Security Symposium, 2013, pp. 589-604.

[32] 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 Proc. 2014 ACM SIGSAC Conference on Computer and Communications Security, 2014, pp. 179-190.

[33] 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, Sep., pp. 2771-2781, 2016.

[34] H. Le, Q. Pham, D. Sahoo, and S.C. Hoi, “URLnet: Learning a URL representation with deep learning for malicious URL detection,” *arXiv preprint arXiv* vol. 1802.03162.

[35] M. He et al., “An efficient phishing webpage detector,” *Expert Systems with Applications*, vol. 38, no. 10, Sep., pp. 12018-12027, 2011.

[36] “Clean-MX,” 2019. [Online]. Available: https://support.clean-mx.com/. [Accessed April. 8, 2019].

[37] “DMOZ” 2019. [Online]. Available: http://dmoz-odp.org/. [Accessed April. 8, 2019].

[38] “Malware Domains List,” 2019. [Online]. Available: https://www.malwaredomainlist.com/. [Accessed April. 8, 2019].

[39] “MalwarePatrol,” 2019. [Online]. Available: https://www.malwarepatrol.net/. [Accessed April. 8, 2019].

[40] “MalwareURL,” 2019. [Online]. Available: https://www.malwareurl.com/. [Accessed April. 8, 2019].

[41] Open DNS, “PhishTank. Out of the Net, into the Tank” 2019. [Online]. Available: https://www.phishtank.com/. [Accessed April. 8, 2019].

[42] “Spam Domain Blacklist,” 2019. [Online]. Available: http://www.joewein.de/sw/blacklist.htm. [Accessed April. 8, 2019].

[43] “VirusTotal” 2019. [Online] Available: https://www.virustotal.com/. [Accessed April. 8, 2019].

[44] Mozilla Developer Network, “HTTP-Headers,” 2018. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/. [Accessed August. 1, 2018].

[45] Python Software Foundation, “Python,” 2018. [Online]. https://www.Python.org/. [Accessed August 10, 2018].

[46] K. Reitz, I. Cordasco, and N. Prewitt, “Requests: HTTP for Humans,” 2019. [Online]. Available: http://docs.Python-requests.org/en/master/. [Accessed August. 10, 2018].

[47] Statsmodels. Statistics in Python, “Variance Inflation Factor,” 2019. [Online]. Available: https://www.statsmodels.org/devel/generated/statsmodels.stats.outliers_influence.variance_inflation_factor.html. [Accessed April. 8, 2019].

[48] 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, Jan., pp. 754-767, 2015.

[49] A.G. Asuero, A. Sayago, and A.G. Gonzalez, “The Correlation Coefficient: An Overview,” *Critical Reviews in Analytical Chemistry*, vol. 36, no. 1, Jan., pp. 41-59, 2007.

[50] UCLA Institute for Digital Education and Research, “What statistical analysis should I use? Statistical analyses using SPSS,” 2019. [Online]. https://stats.idre.ucla.edu/spss/whatstat/what-statistical-analysis-should-i-usestatistical-analyses-using-spss/. [Accessed Febuary. 8, 2019].

[51] S. Kotsiantis, “Supervised Machine Learning: A Review of Classification Techniques,” *Artificial Intelligence Review*, vol. 26, no. 3, Nov., pp. 159-190, 2006.

[52] C. M. Bishop, *Neural Networks for Pattern Recognition*. New York, NY: Oxford University Press, 1995.

[53] R. E. Schapire, “A brief introduction to boosting,” In Proc. the Sixteenth International Joint Conference on Artificial Intelligence, vol. 2, 1999, pp. 1401-1406.

[54] P. Geurts, E. Damien, and L.Wehenkel, “Extremely Randomized Trees,” *Machine Learning*, vol. 63, no.1, Apr., pp. 3-42, 2006.

[55] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, no. 1, Oct., pp. 5-32, 2001.

[56] J. H. Friedman, “Stochastic Gradient Boosting,” *Computational Statistics & Data Analysis*, vol. 38, no. 4, Feb., pp. 367-378, 2002.

[57] L. Breiman, “Bagging predictors,” *Machine Learning*, vol. 24, no. 2, Aug., pp. 123-140, 1996.

[58] P. McCullagh and J. A. Nelder, *Generalized Linear Models*. Boca Raton, FL: Chapman & Hall, 1989.

[59] L. E Peterson, “K-nearest neighbor,” in *Scholarpedia*, vol. 4, no. 2, pp. 1883, 2009. Available: Scholarpedia.org, http://www.scholarpedia.org/article/K-nearest_neighbor [Accessed August 1, 2018].

[60] 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, Jun., pp. 321-357, 2002.

[61] “SMOTE - Synthetic Minority Over-sampling Technique,” [Online]. Available: https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.over_sampling.SMOTE.html. [Accessed February. 8, 2019].

[62] “Scikitlearn,” 2019. [Online]. Available: http://scikitlearn.org/. [Accessed February, 8, 2019].

[63] Featuretools, “An open source Python framework for automated feature engineering,” [Online]. Available: https://www.featuretools.com/. [Accessed February. 8, 2019].

[64] “Correlation and Linear Regression,” 2019. [Online]. Available: http://sphweb.bumc.bu.edu/otlt/MPH-Modules/BS/BS704_Correlation-Regression/BS704_Correlation-Regression_print.html. [Accessed February. 8, 2019].

[65] J. Friedman, T. Hastie, and R. Tibshirani, *The Elements of Statistical Learning*. New York, NY: Springer Series in Statistics, 2009.

[66] R. Kohavi, “A study of cross-validation and bootstrap for accuracy estimation and model selection,” In Proc. 14th International Joint Conference on Artificial Intelligence, 1995, pp. 1137-1145.

[67] Cymon.io, “Open Threat Intelligence,” 2019. [Online]. Available: https://cymon.io/. [Accessed January. 15th, 2019].

TABLE III. MODEL PERFORMANCE WITH AND WITHOUT SAMPLING

Model Performance over Various Scenarios with Prior / All Features															
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 IV. MODEL PERFORMANCE WITH FEATURE TRANSFORMATION

Model Performance over Various Scenarios with Prior / All Features										
Model	*Feature Selection*					*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 V. CROSS-VALIDATION AND HYPERPARAMETER TUNING

Cross-Validation and Tuning			
Model	*Scenario - Split*	*MCC*	*Scoring Metric*
ET	No - 70:30	0.6497	balanced accuracy
KNN	Over - 70:30	**0.6784**	precision weighted
RF	Under - 70:30	0.6414	precision weighted
RF	Feature Selection - 70:30	0.6301	precision micro
BC	PCA - 70:30	0.6773	recall
BC	No - 80:20	0.6764	f1 weighted
KNN	Over - 80:20	0.6722	precision weighted
NN	Under - 80:20	0.6400	recall
BC	Feature Selection - 80:20	0.6249	f1 weighted
BC	PCA - 80:20	0.6713	recall

a. The "/" separates metrics from models built with the 11 previously studied features and models built with the 22 features. The best values in each column are **bold.**