An Image-based Feature Extraction Approach for Phishing Website Detection

Hao Jiang[1], Joshua S. White[1], and Jeanna N. Matthews[1]

[1]Clarkson University

Abstract

Phishing website creators and anti-phishing defenders are in an arms race. Cloning a website is fairly easy and can be automated by any junior programmer. Attempting to recognize numerous phishing links posted in the wild e.g. on social media sites or in email is a constant game of escalation. Automated phishing website detection systems need both speed and accuracy to win. We present a new method of detecting phishing websites and a prototype system LEO (Logo Extraction and cOmparison) that implements it. LEO uses image feature recognition to extract "visual hotspots" of a webpage, and compare these parts with known logo images. LEO can recognize phishing websites that has different layout from the original websites, or logos embedded in images. Comparing to existing visual similarity-based methods, our method has a much wider application range and higher detection accuracy. Our method successfully recognized 24 of 25 random URLs from PhishTank that previously evaded detection of other visual similarity-based methods.

1 Introduction

Phishing is one of the most successful and prominent attack methods[14]. It also requires little technical skill on the part of the attacker. Unlike other attack methods, phishing does not require infiltration of a victim's machine which can leave traces of malicious methods and activity. This alone makes phishing extremely hard to detect. A victim may never know that they are under attack.

Creating a phishing webpage is simple and cheap. If you search Google for the phrase "create a phishing website", there are approximately 208,000 results, most of which are detailed step-by-step tutorials. With a hosting service(or access to compromised machines) and free tools that copy a given URL, anyone can setup a phishing website in minutes. With readily available tools, this process can even be automated.

To deal with phishing websites that are constantly sprouting up all across the Internet, we need an efficient and automated method for identifying them quickly and accurately. As scanning the entire URL address space is impractical, there have been many innovative ideas of where to look for suspicious phishing URLs. A simple but effective way is to ask everyone to report suspicious URLs that they encounter. PhishTank (http://www.phishtank.com) is a website that allows user to submit these suspicious URLs and also to verify the status of URLs submitted by others. Netcraft (http://www.netcraft.com/) offers a Firefox plugin that enables users to report the suspicious URL with a single click. Some researchers are more interested in automating the URL collection work. J. White et al.[19] talk about searching for suspicious URLs in Twitter data. I. Jeun et. al [9] create a honeypot (the "SpamTrap") to collect URLs from spam emails.

Beyond reliable methods for collecting suspicious URLs, we need a fast and reliable method to identify whether these websites are truly phishing pages. This paper presents an innovative image-based feature extraction method for phishing website recognition. Our method works by first taking a screenshot of a target webpage, then locating "visual hotspots" in it. A visual hotspot is a continuous rectangular region that contains non-text visual information. These hotspots represent image features of the target webpage. The features are then compared with the pre-built logo library. If any of these features match a logo in the logo library, the target webpage is thought to be a phishing suspect.

To evaluate our algorithm, we implement a prototype system LEO(Logo Extraction and cOmparison), run it against real world phishing websites, and evaluate its accuracy and performance. Most parts of LEO can be run in parallel, which grants it the scalability necessary for the system to be used for large-scale online phishing detection.

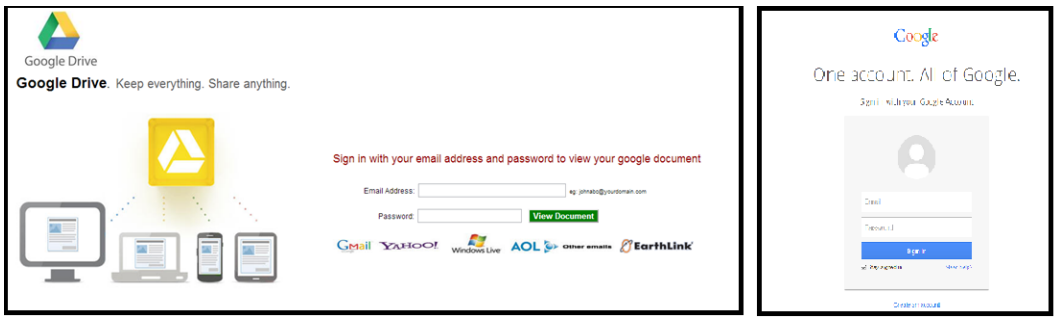

Figure 1. A phishing website of Google Drive

The rest of the paper is organized as follows. Section 2 reviews some previous works of phishing detection, and compares them with our new method. Section 3 describes our new algorithm in more detail and Section 4 presents an overview of our system implementation. Section 5 evaluates the accuracy and performance of LEO. Section 6 presents our conclusion.

2 Previous Work

Different methods have been used to compare the similarity of two webpages and thus used as the basis for identifying phishing websites. We categorize these methods into three high-level types: structure-based comparison, visual-based comparison and content-based comparison.

Structure-based comparison works under the assumption that similar webpages will have a similar underlying DOM tree structure. Structure-based comparison methods first parse the HTML webpages to construct the corresponding DOM trees, and then use various algorithms to compare them. Rosiello et al. [15] compare the HTML tags in the DOM tree, looking for similar sub-tree structures. This method is effective against phishing websites that directly copy the content of original websites. However it has an obvious disadvantage. The appearance of a webpage cannot be uniquely defined by its DOM tree structure. Thus an attacker can easily avoid such detection by using different HTML tags to generate webpages that look similar. For example, using <DIV>instead of <TABLE>element to do page layout can lead to exactly the same visual effect, while maintaining a totally different DOM Tree. In addition, attackers can choose to dynamically generate DOM trees via scripts at runtime. In this case, directly accessing the webpage URL cannot get the DOM tree required, which will lead to false negatives.

Visual-based comparison on the other hand focuses on the final visual effect rather than on the underlying DOM structure. Specifically, the screenshots of webpages are captured using techniques such as a headless browser. Comparison between these images are then conducted using various image processing techniques. Visual comparison methods overcome some of the disadvantage of DOM-based methods by focusing on comparing the visual output of a webpage, which is the exact same image seen by the end users.

A.Fu et al. talk about their work [7] of using Earth's Mover Distance (EMD),[16] to evaluate the difference between two webpages. EMD is a metric of the similarity between two probability distributions over a region. The closer two images are, the smaller the EMD value will be. J.White et al. talk about using the hash value of images in their work[19]. The authors calculate pHash of a screenshot and evaluate the difference using hamming distance between two hash values. They present experimental results illustrating that adding a small change to the original image will also lead a small increase in the hamming distance.

Bohunsky et al.[3] describes their work of using the visual image to do webpage comparison and clustering. Instead of comparing the entire picture, they split the webpage into small rectangular areas which they call "visual boxes". By comparing the visual box structure of two webpages, they attempt to detect the correlation. However, two webpages that have a similar layout but totally different text content may be categorized as similar. For example, almost all news website present news with a title picture and an abstract. Despite the visual similarity, the content they talk about may be completely different. This shows that relying only on visual layout of a webpage for clustering may introduce a high rate of false positives.

These works are efficient against phishing websites that look exactly the same as the original websites. However, in practice we have found some phishing examples that do not like the original websites at all. Thus they evade this kind of detection easily. Figure 1 shows an example phishing website of Google Drive we retrieved from PhishTank, as well as the actual Google Drive login page. We can see that this phishing webpage does not actually look like the

original legitimate Google Drive page. We have observed over 200 samples from PhishTank and noticed that this is not a single rare case.

We believe that ”low-quality” phishing webpages, i.e. those that look very different from the original source, still have a possibility of catching victims. Certainly, not all the web-surfers have enough knowledge to compare the phishing webpage to the legitimate one. This is especially true if victims are not familiar with the actual webpage they are intending to access. Failing to catch this type of phishing websites is a big disadvantage of the discussed methods.

G. Wang et al described an idea that is closest to our effort in [18]. Noticing the importance of logos in detection of phishing website, they create a Firefox plugin named Verilogo that is capable of extracting image files from `<IMG>` tags in webpages. The extracted images are then compared with known website logos using SIFT[12] method. If a webpage contains images that are identical to some known logos and the webpage is not authorized to use the logo, this webpage is flagged as a phishing suspect. In the case that phishing websites use logos as separate image files, we believe that this method works as good as ours. However, this method fails to deal with “embedded” logos, in which case logo images are included as part of the background image, as what we can see in Figure 1. The new method we propose is capable of extracting logos from the background image, thus overcoming this problem.

Content-based comparison focuses on comparison of the webpage text, often using machine learning techniques. Y. Zhang et al.[21] create a content-based phishing website detection system in which they apply TF-IDF[10], an algorithm that is widely adopted in text mining and information retrieval, to the webpage text. R. Basnet et al. [2] apply different machine learning techniques, including SVM, neural networks and SOM and evaluate their performance. C. Whittaker et al.[20] present the automatic maintenance of Google’s blacklist of phishing websites with similar feature sets using classification.

S. Abu-Nimeh et al.[1] present a performance comparison of different methods for detection of phishing email. D. Miyamoto [13] did similar work on phishing website detection and showed AdaBoost[6] works best in their particular case.

Content-based phishing detection also has the advantage of performance when being compared to image-based methods. However, content-based methods typically have a higher error rate and higher false positive rate, which limits their accuracy. We believe that the combination of both content-based feature and image-based feature, will allow us a better result. This is fundamental to our future research interests.

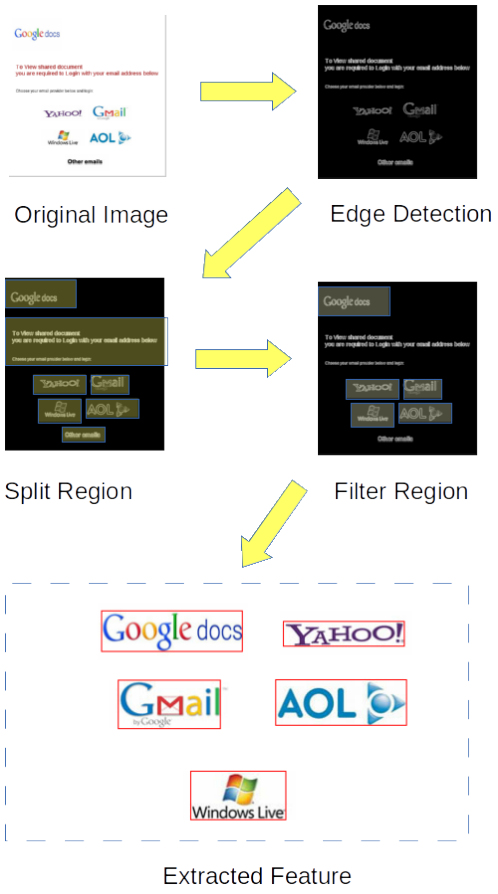

Figure 2. Feature Extraction Process

3 Algorithm Description

In this section, we describe the details of our new method of visual-based feature comparison. We first focus on the extraction of ”features”, or prominent visual elements from a webpage. Our goal is to isolate recognizable logos. We observe that most phishing websites, even ”low-quality” ones that do not look like the original webpage, will at least include a logo. Thus this provides us a perfect target for phishing detection.

3.1 Feature Extraction

To extract features, we first distinguish valid graphical information from background image using edge detection, then split it into small rectangular regions. We then apply a collection of filters to the region set in order to identify the features most likely to contain a logo. Figure 2 shows this process. We describe each step in detail in the following paragraphs.

Edge detection is done by calculating the gradient of

each pixel based on its 3x3 neighbors. We first calculate the horizontal and vertical gradients using central difference vector $\mathbf{t} = [-1, 0, 1]^T$. Thus we have

$$\nabla_x(x, y) = f(x + 1, y) - f(x - 1, y)$$
$$\nabla_y(x, y) = f(x, y + 1) - f(x, y - 1)$$

We then calculate the gradient value at point (x, y) as

$$\nabla = \frac{1}{\sqrt{2}}\sqrt{(\nabla x)^2 + (\nabla y)^2}$$

The result is then rounded to an integer between 0 and 255. For an RGB image, we repeat the calculation for the three different colors and get the biggest value as the final result.

Edge detection is good at removing constant background color. However, some webpages use a background of image gradient, which causes ∇ at the background region a small non-zero value. To remove such interference, we setup a threshold T and write h(x,y) as a piece-wise function.

$$h(x, y) = \begin{cases} \nabla & : \nabla \geq T \\ 0 & : \nabla < T \end{cases}$$

By properly choosing the value of T, we make sure ∇ at background will be 0, which gives us a grayscale image that we used as the input for region splitting.

The goal of region splitting is to separate the image into smaller regions that contain non-black pixels. Given a region to be split, we first calculate the lower bound of this region, i.e. the smallest rectangle that encloses all the non-black pixels in this region. We then try to draw a line to split this region, in either the vertical or horizontal direction. There may be multiple possible lines, and we choose the one that gives a maximal margin. If such a line can be found, we split the given region into two sub-rectangles, and then repeat the process on each of these sub-rectangles. If no line can be found, we switch to rectangular splitting. This method of rectangular splitting tries to find the biggest sub-rectangle in the given region. If none of the previous methods work, we consider the region as unsplittable and add it to the result list. These unsplittable rectangles are input to the filtering step. Figure 3 shows the pseudo-code of the algorithm we used for region splitting.

We then apply a set of filters to split regions. These filters are designed to remove regions that are unlikely to contain logos or other important features. First, we observe that it would be difficult for a region that is too small or too narrow to contain any valid information. One example is the rectangle that encloses a horizontal rule created by
tag. We setup a threshold T for the dimension of the rectangles, and ignore all those have a height or width less than T.

Second, we prefer filter out regions that contain text rather than images. We observe that when users first see a webpage, the first thing that catches their eyes is image

```
function SPLIT(Rectangle region, List result)
    ;; Remove excessive space
    LOWERBOUND(region);
    ;; First try to split the region using lines
    vline ← VLINE(region);
    hline ← HLINE(region);
    line ← MAXMARGIN(vline, hline);
    if line ≠ NULL then
        region1, region2 ← LINESPLIT(region, line);
        SPLIT(region1, result);
        SPLIT(region2, result);
        return ;
    end if
    ;; Split the region use rectangle
    region ← RECTSPLIT(region);
    if region ≠ NULL then
        SPLIT(region,result);
        return ;
    end if
    ; Not splittable, add the region to result list
    result.ADD(region);
    return
end function
```

Figure 3. Algorithm of Region Splitting

rather than text. We have trained a SVM model to identify regions that contain only text. To do this, we notice that the vertical distribution of a character image shows an interesting pattern. More specifically, consider that when we write on a ruled piece of paper, only character "j, g, y" will occupy the lower part of the text region, and only "h, i, j" will occupy the higher part. Most of the characters are at the center part. Thus if we calculate the percentage of non-black pixels on each row of a text region, we can expect to get a consistent pattern which can be learned by SVM. Figure 4 shows this distribution pattern. This algorithm is specific to text written in a Latin based alphabet but we are interested in developing similar filters for other alphabets such as Cyrillic, Arabic or Asian characters.

To deal with fonts of different sizes or heights, we first scale the candidate region to a parameterized height H, preserving the ratio of width and height. Next, for each row of pixels, we calculate the percentage of non-black pixels. This gives us a histogram of H bins, which is then used as feature descriptor to do classification after being normalized.

Our splitting method works well with most of the images, extracting image parts from their background. However, we have a problem of over-splitting, in which a region that should be kept together is split incorrectly. For example, Google's logo has big vertical gap between "G" and

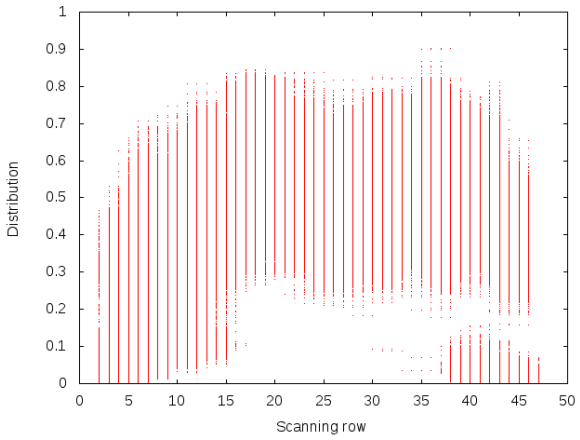

Figure 4. Text-image row scanning distribution pattern

the first "o". This logo will be split by our algorithm if the resolution is too high.

To solve this problem, we add a step that combines these over-split regions into a whole. We balance the tension between over-splitting and over-combining in this way. First, we have a threshold for how close the two regions must be in order to combine them. Second, we check that the combination will not introduce too much whitespace. We setup a threshold for the percentage of newly introduced whitespace that is allowed. With these rules, we will first group the regions, then for each such group, draw a rectangle to cover it as a combined result.

3.2 Feature Comparison

Now that we have covered our algorithm for feature identification, we are ready to move on to discussing the method we use for comparing features. Restating the problem to be solved, given the image features extracted from a suspicious webpage, we want to know whether they are identical to known logos.

Our method is based on SVM classification using HOG descriptor. In [5], N. Dalal and B. Triggs propose the Histogram of Oriented Gradient (HOG) feature descriptor of image, for the purpose of object detection. It had been proved extremely effective in human facial recognition. In [17], Shrivastava et. al. demonstrate the effectiveness of using HOG to do clustering of pictures with natural scenes.

Recall that in previous section we described how we calculate the gradient value $\nabla(x, y)$ of a given point (x, y). The result we get there is a scalar. In the HOG processing, we also take the direction of the gradient into account and get a normalized vector.

$$\vec{\nabla}(x, y) = \frac{1}{\sqrt{2}}[\nabla_x, \nabla_y]$$

We split the image into $m \times n$ grids. Each cell of the grid is a $k \times k$ square. For each cell of the grid, we calculate the gradient vector of each pixels in that cell, which is ∇_1 to ∇_{k^2}. These k^2 vectors are separated into w bucket based on their angles. For example, when w is 4, we have four buckets that contains vectors with angles in $[0, \frac{\pi}{2})$, $[\frac{\pi}{2}, \pi)$, $[\pi, \frac{3\pi}{2})$ and $[\frac{3\pi}{2}, 2\pi)$. This forms the HOG descriptor of that cell.

For each cell, we sum the value in each bucket up and normalize them to get w values, and for the entire image, we get $m \times n \times w$ values. These values form the descriptor we use for SVM classification. The size of the HOG descriptor is proportional to the image size given a fixed cell size, thus to compare image features of different size. We need to scale them to a predefined fixed dimension $[M, N]$.

To correctly match image features that contain the same content, but are different in size, we use a scale-invariant method. We prepare the training data by first stretching the source image to different dimensions, then scaling them all back to dimension $[M, N]$ to generate positive training samples. We then use the target image to generate the test set. From the classification result we can tell whether two image features are the same.

To detect phishing websites based on logo recognition, we first prepare a logo library that contains collected logo images of commonly phished sites, then train a multi-class SVM classifier based on features generated from these images. Each logo image is represented by a separated class in this classifier. If a test image falls into any of these classes, we consider the webpage that contains this image a candidate of phishing website. We emphasize that any organization could prepare a logo library specific to their interests(e.g. looking for any logos they use in their legitimate media campaigns).

Category	Name	Value
Filter	Minimal Width	5px
Filter	Minimal Height	5px
Filter	Minimal Area	100px^2
Filter	Height Threshold	25px
SVM	Dimension	$500\times500\text{px}^2$
SVM	HOG Bucket Size	9
SVM	HOG Cell Size	50px
SVM	Height	50px

Table 1. Parameter value

4 Implementation

In this section, we present the implementation of LEO, including our settings of tunable parameters, training data used, the logo library we used, the environment in which we have run our experiments and the external tools we have used.

The main program is written in Java and Javascript, containing around 4000 lines of code. All the source code is readily available on our website. We use PhantomJS[8] for webpage screenshot image retrieval and Libsvm[4] for SVM classification.

4.1 Parameter values

In this section we explain the parameter values used in our algorithm. Table 1 lists relevant parameters and their current values.

The first three parameters control the behavior of the size filter. Any region that either has its width/height less than 5px or has an area less than $100px^2$ is considered unable to hold valid information and is thus ignored.

In order to decrease the false positive in text identification, we introduce a height threshold for text detection. With our observation, most webpages have a text font size between 10 and 16px. With a given candidate, we will first try to split it into rows, and apply text filtering SVM to each row. If a region cannot be split horizontally and it has a height bigger than 25px, we think it is not a normal text and skip the text filter step.

The next two parameters are for the HOG descriptor. As described in previous section, the size of HOG descriptor is proportional to the size of the input image. Thus in order to do comparison between different images, we need to first scale them to the same size. Setting the size too large will lead to a bigger descriptor and affect the performance, while setting the size too small will lead to a substantial information loss and impact the accuracy. We tried different dimensions(1024×768, 800×600, 500×500) with the same cell size ($50\times50px^2$) under some common dimensions, and choose the smallest dimension while maintaining the accuracy.

Similarly, we tried different sizes for HOG cells, and chose the biggest one that does not impact the performance severely. In [5], Dalal and Triggs suggest using unsigned gradients and set the bucket size to 9, which their experiments suggest offers the best performance for detection of human faces. Here we start with the same setting and get a satisfying result. Thus we kept this parameter unchanged.

The last parameter we want to mention here is the height H used by SVM to identify the text region. In this case, the model can be pre-trained and the performance is not a big problem. A small descriptor length will affect the accuracy. We double the threshold value for text height in text filtering, which gives us 50.

4.2 SVM training data

Our methods relies on SVM classification heavily to identify the text region as well as compare feature images, which requires a sufficient large training set to function properly.

For text identification, we generate positive samples by creating images that contains strings with random length, font and size. We use four most common fonts: Geogia, Sans-Serif, Arial and Courier. The font size varies between 12 and 20, which we believes covers most common font size in webpages. We use both text from books(e.g. the Bible) and random words from dictionary to generate 45541 positive samples. For feature image comparison, we scale the feature image under comparison into 100 different dimensions, from 100×100 to 1100×1100. These scaled images are then used to generate positive training samples for recognizing this image, which means each model are trained with 100 positive samples.

We also need some random picture data to be used as negative training data. We gather these data from Flickr (http://www.flickr.com/). Flickr is a picture sharing image that provides a place for people around the world to upload and share the picture they shoot. This makes it a perfect place to retrieve random pictures. Flicker does not provide a function for downloading packed pictures, but we used PhantomJS to automatically download pictures from Flickr photo streams. We repeat this process and collect 10843 unique pictures. They are used as negative training samples for image feature comparison. The Javascript we used to download these pictures can also be found in our source code.

4.3 Logo library

According to Kaspersky Lab's technical report [11] about phishing attacks in 2013, over 90% phishing attacks target at social networks, financial services and mail services. Besides these, we also notice a trend of attacks against personal cloud service providers. As a demo system, we choose 15 logos from top companies in these fields. The detail list is provided in Table 2. The process of comparing a given feature to these logos is fully parallelized. Thus in a production system, unlimited number of logos can be added to this library without affecting system performance. We will talk more about this when discussing the performance of our system.

Field	Logo
Financial System	eBay, Amazon, PayPal, HSBC, IRS, BOA
Social Network	Facebook, Twitter, LinkedIn
Mail Service	Gmail, Outlook, Yahoo!
Cloud Service	Google Drive, Dropbox, Box

Table 2. Logo Library content

5 Experimental Results

We conduct our experiment on a test machine with AMD A10-6800K quad-core APU and 8GB memory, installed with Ubuntu Desktop 13.10 64-bit and Oracle JDK 1.7.0_45 64-bit for Ubuntu. All test screenshots are captured in 1920x1080 resolution.

5.1 Accuracy of Text Filtering and Image Comparison

We first present the accuracy of our text filtering algorithm. We have prepared two test sets: a positive test set by generating images that each contains a text string from English literature. Our algorithm is considered successful if it can recognize these images as "image contains only text", In other words, it answers "yes" to these images. We then prepare a negative test set using pictures downloaded from Flicker website. Comprising mostly of landscape and portrait photos, these images are not likely to contain only text strings. Thus we expect our algorithm to answer "no" to them.

We prepare a positive test set that contains 9468 images, each containing a single row of text of 80 characters. All the sentences are extracted from Leo Tolstoy's *Peace and War*. The negative test set we use contains 11942 pictures downloaded from Flickr. The test result is shown in Table 3. It can be seen that our text-filtering algorithm have a false-positive rate of 0.831% and a false-negative rate of 0.01%. This is a convincing result that shows our text-filtering algorithm has a high accuracy of detecting images that contain only text.

To address image comparison accuracy, our test has been designed as follows: we randomly choose a picture from Flicker and insert it into LEO's logo library, then run LEO on a test input that contains both the scaled original picture and other irrelevant pictures. The test is thought to be successful if LEO can recognize the original picture. We repeat the test 200 times and always get 100% accuracy. This is a strong support to the effectiveness of our algorithm.

Type	Input	Correct	Accuracy
Text Image	9468	9465	99.97%
Non-text Image	11942	11764	98.51%

Table 3. Text Identification Accuracy

5.2 Application to Phishing Websites

We manually choose 25 URLs fetched from PhishTank as our test set. These URLs are chosen with following guidelines: the phishing webpage contains the logo of the original website; the logo is included in our logo library; the visual appearance of the phishing webpage is different from the original one. All previous detection methods that based on visual features are not able to deal with these phishing websites. As a result, we successfully identify 24 out of 25 test cases as phishing websites. The only case that failed is because that the logo is partially overlapped by a floating layer on the webpage, but we still are about to extract a rectangle that enclose the logo.

This test also shows the improvement of our methods comparing to existing visual similarity-based methods. In Figure 5, we show a phishing website of PayPal that does not look like the real one. In addition, the logo is not a separate image file but embedded in background image. Nevertheless, our method successfully located the PayPal logo in this case, which is the region marked by red rectangle, and thus detect this phishing website. This shows that our method overcomes the limitation of existing visual similarity-based method and is immune to webpage layout change – as long as the phishing websites use the logo of the original website, which we believe they surely will do – our method can always locate the logo and thus detect the phishing website.

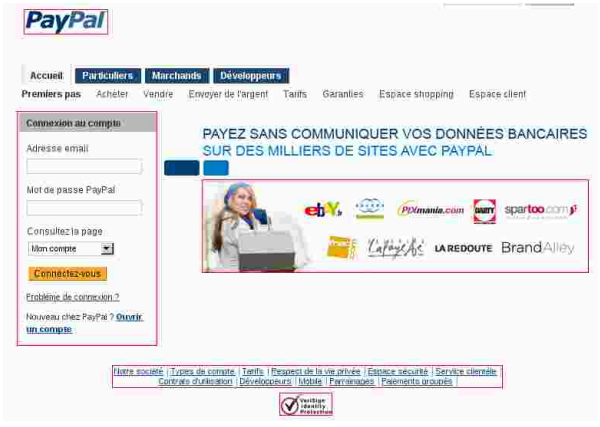

Figure 5. Detection of PayPal phishing

5.3 Performance Analysis

In this section we present the performance test result of LEO. Running on the test machine, the time required to identify one webpage of size 1920x1080 is in average 3.11 seconds, with maximal value 9.7 seconds and minimal value 2.1 seconds. We noticed that the time required for image extraction is primarily related to the complexity of page layout.

In a production system, we can increase the throughput by simply adding more machines to process different webpages in parallel. In addition, the performance can be further increased by parallelize some of the steps. As we adopt a top-down method when splitting the region, split regions do not overlap and can be processed in parallel. As an example, we have provided a parallel version of split/combine operations in LEO's source code.

The process of logo recognition can also be done in parallel. By preparing different logo library and distribute them to different machines in a cluster, an image feature can be compared with multiple logos concurrently. This allows the system to have full scalability with the size increasing of the logo library. We have included a simple distributed framework in LEO's source code that allows users to build a cluster to process logo recognition. This framework can be easily extended to support other distributed frameworks such as Apache Hadoop.

6 Conclusion

In this paper we presented a new image-based feature extraction method for phishing website detection. We also showed a prototype system LEO that implements the algorithm. LEO is able to recognize phishing websites by extracting logos from webpage screenshots, which make it immune to attacks like layout changing or logo embedding. We apply LEO to real-world phishing examples and obtain evaluation results. Our results show that LEO is able to deal with different types of phishing, while maintaining high accuracy and scalable performance. We believe that LEO can be used in conjunction with existing content-based method to further increase the accuracy of phishing detection.

References

[1] S. Abu-Nimeh, D. Nappa, X. Wang, and S. Nair. A comparison of machine learning techniques for phishing detection. In *eCrime '07*, pages 60–69. ACM, 2007.

[2] R. Basnet, S. Mukkamala, and A. Sung. Detection of phishing attacks: A machine learning approach. In *Soft Computing Applications in Industry*, pages 373–383. 2008.

[3] P. Bohunsky and W. Gatterbauer. Visual structure-based web page clustering and retrieval. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 1067–1068, New York, NY, USA, 2010. ACM.

[4] C.-C. Chang and C.-J. Lin. Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2(3):27:1–27:27, May 2011.

[5] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR 2005*, volume 1, pages 886–893 vol. 1, 2005.

[6] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119 – 139, 1997.

[7] A. Y. Fu, L. Wenyin, and X. Deng. Detecting phishing web pages with visual similarity assessment based on earth mover's distance (emd). *IEEE TDSC*, 3(4):301–311, 2006.

[8] A. Hidayat. PhantomJS library. http://phantomjs.org/, 2010–2014.

[9] I. Jeun, Y. Lee, and D. Won. Collecting and filtering out phishing suspicious urls using spamtrap system. In *Grid and Pervasive Computing*, volume 7861, pages 796–802. 2013.

[10] K. S. Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28:11–21, 1972.

[11] K. Lab. Financial cyber threats in 2013. Technical report, Kaspersky Lab, Apr. 2014.

[12] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, Nov. 2004.

[13] D. Miyamoto, H. Hazeyama, and Y. Kadobayashi. An evaluation of machine learning-based methods for detection of phishing sites. In *Advances in Neuro-Information Processing*, volume 5506, pages 539–546. 2009.

[14] T. Moore and R. Clayton. Examining the impact of web site take-down on phishing. *eCrime '07*, pages 1–13, 2007.

[15] A. P. E. Rosiello, E. Kirda, C. Kruegel, and F. Ferrandi. A layout-similarity-based approach for detecting phishing pages. In *SecureComm 2007*, pages 454–463, 2007.

[16] Y. Rubner, C. Tomasi, and L. Guibas. A metric for distributions with applications to image databases. In *Computer Vision, 1998. Sixth International Conference on*, pages 59–66, 1998.

[17] A. Shrivastava, T. Malisiewicz, A. Gupta, and A. A. Efros. Data-driven visual similarity for cross-domain image matching. *ACM Transaction of Graphics (TOG) (Proceedings of ACM SIGGRAPH ASIA)*, 30(6), 2011.

[18] G. Wang, H. Liu, S. Becerra, K. Wang, S. Belongie, H. Shacham, and S. Savage. Verilogo: Proactive phishing detection via logo recognition. Technical Report CS2011-0969, UC San Diego, Aug. 2011.

[19] J. S. White, J. N. Matthews, and J. L. Stacy. A method for the automated detection phishing websites through both site characteristics and image analysis. In *SPIE '12*, 2012.

[20] C. Whittaker, B. Ryner, and M. Nazif. Large-scale automatic classification of phishing pages. In *NDSS '10*, 2010.

[21] Y. Zhang, J. Hong, and L. Cranor. CANTINA: A content-based approach to detecting phishing web sites. 2007.