2018 IEEE Symposium on Security and Privacy Workshops

Detecting Homoglyph Attacks with a Siamese Neural Network

Jonathan Woodbridge, Hyrum S. Anderson, Anjum Ahuja, Daniel Grant
Endgame
Email: {jwoodbridge, hyrum, aahuja, dgrant}@endgame.com

***Abstract*—A homoglyph (name spoofing) attack is a common technique used by adversaries to obfuscate file and domain names. This technique creates process or domain names that are visually similar to legitimate and recognized names. For instance, an attacker may create malware with the name *svch0st.exe* so that in a visual inspection of running processes or a directory listing, the process or file name might be mistaken as the Windows system process *svchost.exe*. There has been limited published research on detecting homoglyph attacks. Current approaches rely on string comparison algorithms (such as Levenshtein distance) that result in computationally heavy solutions with a high number of false positives. In addition, there is a deficiency in the number of publicly available datasets for reproducible research, with most datasets focused on phishing attacks, in which homoglyphs are not always used.**

This paper presents a fundamentally different solution to this problem using a Siamese convolutional neural network (CNN). Rather than leveraging similarity based on character swaps and deletions, this technique uses a learned metric on strings rendered as images: a CNN learns features that are optimized to detect visual similarity of the rendered strings. The trained model is used to convert thousands of potentially targeted process or domain names to feature vectors. These feature vectors are indexed using randomized KD-Trees to make similarity searches extremely fast with minimal computational processing. This technique shows a considerable 13% to 45% improvement over baseline techniques in terms of area under the receiver operating characteristic curve (ROC AUC). In addition, we provide both code and data to further future research.

***Keywords*-deep learning; siamese networks; homoglyph;**

I. Introduction

Cyber attackers have long leveraged creative attacks to infiltrate networks. One simple attack uses homoglyphs or name spoofing to obfuscate malicious purpose. These attacks occur for both domain names and process names. Attackers may use simple replacements such as *0* for *o*, *rn* for *m*, and *cl* for *d*. Swaps may also include Unicode characters that look very similar to common ASCII characters such as *ł* for *l*. Other attacks append characters to the end of a name that seem valid to a user such as *svchost32.exe*, *svchost64.exe*, and *svchost1.exe*. The hope is that these processes or domain names will go undetected by users and security organizations by blending in as legitimate names.

One naive approach for discovering name spoof attacks is to calculate the edit (Levenshtein) distance of each new process or domain name to each member of a set of processes or domain names to monitor. In general, edit distance measures the number of *edits* (insertions, deletions, substitutions or transpositions) to convert one string to another. A distance less than or equal to a pre-defined threshold is flagged as a potential spoof. In practice, this approach suffers from a poor False Positive (FP)/False Negative (FN) tradeoff.

Another approach is to create a custom edit distance function that accounts for the visual similarity of substitutions, so that substituting a character with a visually similar character result in a smaller edit distance than a visually distinct character [15], [4]. As shown later in the paper, these techniques result in only modest improvements over standard edit distance functions. In addition, these techniques are in large part manually crafted, making them very difficult to enumerate and maintain, especially when considering the full Unicode alphabet.

To overcome the shortcomings of the aforementioned methods, this paper presents a metric-learning technique based on a Siamese convolutional neural network (CNN). A training set $\{(s_i, s'_i, y_i)\}_{i=1}^{n}$ is composed of n pairs of strings consisting of either process names or domain names, together with a distance target (similarity label) y_i. A pair of strings (s_i, s'_i) for which s'_i is a spoof of s_i (or vice versa), we assign $y_i = 0$ (similar), and $y_i = 1$ (dissimilar) otherwise. Each string s_i and its pair s'_i is then rendered as a binary image $\mathbf{x}_i$ and $\mathbf{x}'_i$, respectively. The Siamese CNN is explicitly trained to convert images to features vectors such that the distance between feature vectors of spoofing pairs target a distance of 0.0, and at least 1.0 otherwise. The model is deployed as a defensive measure as follows. We convert all common or potentially targeted domain or process names to feature vectors. These feature vectors are indexed using a randomized KD-Tree index. When a new process or domain name is observed, it is converted to a feature vector by the CNN and searched in the KD-Tree index to find any visually similar matches. If a match exists, then a homoglyph attack is detected.

On the surface, this problem may seem similar to other well studied problems. For example, there is a large body of work that addresses the discovery of phishing attacks [16]. Often these attacks are waged via email so as to trick unsuspecting victims to click on malicious domain names that appear to be benign in order to steal information. Despite some similarities, much of the work in phishing detection is largely not applicable to detecting homoglyph attacks. First, phishing attacks often use domains that appear

DOI 10.1109/SPW.2018.00012

to be legitimate, but are visually distinct from the benign domain that they are impersonating. For example, an attacker may register the domain *google-weekly-updates.com*. In this example, the domain is very different from *google.com* and probably unlikely to be registered (at least not at the time of this publication!). In fact, previous works found that the likelihood of a phishing attack grows with increasing edit distance between the phishing domain and the legitimate domain [16]. Second, phishing detection can use contextual information such as appearance of the web pages (e.g., does the content of *fake-facebook.com* appear legitimate?) and whois information (i.e., registration information related to the domain name).

There has also been a large amount of work in regards to finding nearest string matches in other domains such as data cleansing, spell checking, and bioinformatics [8], [23], [7]. However, those works did not consider visual similarity of characters and do not apply to the problem at hand. Instead, this work is largely inspired by the work in [11] that uses a similar Siamese network used in this paper to classify digits in the MNIST [13] and NORB [14] datasets.

Our work makes three primary contributions:

1) Presents a generic name spoofing detection technique using a Siamese CNN, which to our knowledge, is the first such application of metric learning to homoglyph detection,
2) Compares the system's efficacy to other common string comparison techniques, and
3) Contributes source code to reproduce results in this paper as well as two datasets to further research in this area[1].

In Section II we take a deeper dive into related work as well as the motivations behind this work. In Section III we discuss the high-level design of the system and the architecture of our neural network. In Section IV we compare our neural network to other string matching techniques. We conclude the paper in Section V with some closing thoughts.

II. Related Work

An extensive amount of work has been devoted to efficient string matching. Some of this work is focused on making string matching fast [7], [8] while other work focuses on improving the quality of nearest neighbor searches [23]. However, conventional string matching algorithms are not an effective technique for detecting name spoofing. For example, consider the windows application *iexplore.exe*. A malicious user may create a piece of malware with the name *iexp1orc.exe* that is an edit distance of 2 from the original executable. In this case, a system that labels all process names with an edit distance of 2 or less would catch this spoof attack. However, consider also the common windows process *explorer.exe*. This process is also an edit distance 2 from *iexplore.exe* resulting in a false positive.

[1]https://github.com/endgameinc/homoglyph

The key to detecting name spoofing attacks is to make visual comparisons. When visually comparing the three strings *iexplore.exe*, *iexp1orc.exe*, and *explorer.exe*, one of the strings looks very different from the other two. The first and last character (before *.exe*) of the string *explorer.exe* has a different shape from the other two strings making it very distinguishable. Such distinguishable characters are unlikely to fool anyone in a spoofing attack, but are lost in basic string matching systems.

There are many subtle string updates that result in a string that appears almost identical to the original string. In addition, the Windows operating system supports Unicode characters resulting in an exponentially large number of string swaps making signature-based detection infeasible (i.e., a lookup table of all possible character swaps). Several spoofing attempts are given in Table I. Notice how easily spoofing strings may be overlooked. Authors in [22] give more in-depth analysis of characteristics that make strings appear visually similar.

Table I
Example of process name homoglyphs

Original	Spoof	Edit Distance
SVCHOST.EXE	*SVCH0ST.EXE*	1
LSASS.EXE	*LS4SS.EXE*	1
iexplore.exe	*iexp1orc.exe*	2
chtime.exe	*chtirne.exe*	2

Authors in [4], [15] attempted to improve upon conventional edit distance functions by adding knowledge of visual likeness in characters. For example, swapping a *r* for a *n* would result in a smaller distance than a *y* or a *b*. This technique relied on a largely manual step of deriving similarity measures between characters and did not include the massive unicode set. While this method generally improves upon conventional techniques, it still exhibits a high false positive rate.

A. Phishing

Phishing attacks can be broken down into four categories [9]:

1) Obfuscating a domain name with an IP address,
2) Obfuscating a domain name with another domain name,
3) Obfuscating a domain name within a longer domain name, and
4) Obfuscating a domain name using misspellings and common typos.

Examples of each type of phishing attack is given in Table II. The first three obfuscation techniques result in domains that are not visually similar. While the target domain name may be a substring of the phishing domain name, the two strings are visually different. The fourth obfuscation technique may seem to be similar to process name spoofing, however, misspellings and typos are not necessarily visually similar.

Table II
EXAMPLE OF URL PHISHING ATTACKS. ONLY THE LAST IS A HOMOGLYPH.

Type	Phish URL	Target Domain
1	*202.0.0.1/google.com*	*google.com*
2	*badDomain.com/google.com*	*google.com*
3	*google.com.badDomain.com*	*google.com*
4	*google.om*	*google.com*

Machine learning based approaches for detecting phishing domains rely on two types of features [9], [26], [2], [25], [18]. These include *domain-based features* that are derived directly from the domain name and *page-based features* that are derived from the hosted page.

These techniques have been effective in phishing detection, however, they do not focus on visual similarity. In fact, authors in [16] found that the likelihood of a phishing attack grows with increasing edit distance between the phishing domain and the legitimate domain. Thus, methods to detect phishing attacks are largely not applicable to detecting spoofing attacks. While a new set of features could be derived to detect name spoofing, this process is extremely time consuming and highly susceptible to the cat and mouse games waged by adversarial actors. For these reasons, this work focused solely on visual appearance and relies on convolutional neural networks to derive its own visual features.

B. Siamese Neural Networks

Siamese neural networks were first introduced in 1993 by Bromely and LeCun as a method to validate handwritten signatures [5]. At its core, a Siamese neural network is simply a pair of identical neural networks (i.e., shared weights) which accept distinct inputs, but whose outputs are merged by a simple comparative energy function. The key purpose of the neural network is to map a high-dimensional input (e.g., an image) into a target space, such that a simple comparison of the targets by the energy function approximates a more difficult-to-define "semantic" comparison in the input space.

Mathematically, if a neural network $\mathbf{g_W} : \mathbb{R}^n \mapsto \mathbb{R}^d$ is parameterized by weights $\mathbf{W}$, and we choose simple Euclidean distance for our comparative energy function $E : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$, then the Siamese network computes dissimilarity between the pair of images $(\mathbf{x}_1, \mathbf{x}_2)$ simply as

$$\begin{aligned} d_{\mathbf{W}}(\mathbf{x}_1, \mathbf{x}_2) &= E\left(\mathbf{g_W}(\mathbf{x}_1), \mathbf{g_W}(\mathbf{x}_2)\right) \\ &= ||\mathbf{g_W}(\mathbf{x}_1) - \mathbf{g_W}(\mathbf{x}_2)||_2. \end{aligned} \quad (1)$$

Note that $\mathbf{g_W}$ represents a family of functions parameterized by $\mathbf{W}$. We wish to learn $\mathbf{W}$ such that $d_{\mathbf{W}}(\mathbf{x}_1, \mathbf{x}_2)$ is small if $\mathbf{x}_1$ and $\mathbf{x}_2$ are similar, and large if they are dissimilar. At first glance, one may be tempted to choose $\mathbf{W}$ simply minimizing $d_{\mathbf{W}}$ over pairs of similar inputs; however, this may lead to degenerate solutions such as $\mathbf{g_W} = \text{constant}$, for which $d_{\mathbf{W}}$ is identically zero. Instead, previous research has employed *contrastive loss* to ensure that similar inputs result in small $d_{\mathbf{W}}$, while simultaneously pushing $d_{\mathbf{W}}$ to be large for dissimilar inputs [6].

Chopra et al. [11] proposed a contrastive loss function of the form

$$\mathcal{L}(\mathbf{W}) = \sum_{i=1}^{P} (1 - y_i) L_S\left(d_{\mathbf{W}}^i\right) + y_i L_D\left(d_{\mathbf{W}}^i\right), \quad (2)$$

where $y_i = 0$ if the images in the ith input pair $(\mathbf{x}_1, \mathbf{x}_2)^i$ are deemed similar and $y_i = 1$ if dissimilar, $d_{\mathbf{W}}^i = d_{\mathbf{W}}\left((\mathbf{x}_1, \mathbf{x}_2)^i\right)$ is the Siamese network dissimilarity for the ith pair, and the summation occurs over all P input pairs. The authors chose partial loss for similar pairs to be squared loss, $L_S(x) = x^2$, while partial loss for dissimilar pairs was chosen to be the squared hinge loss with margin α, $L_D(x) = \left(\max\{0, \alpha - x\}\right)^2$. Intuitively, this loss aims to shrink the distance between feature vectors of similar pairs to 0, while expanding the distance between dissimilar pairs to be at least α. In our experiments, we use a margin of $\alpha = 1$.

Since the loss function is differentiable with respect to $\mathbf{W}$, the weights can learned via backpropagation. Notable is the fact that after the weights $\mathbf{W}$ have been trained, the network $\mathbf{g_W}$ may be used in isolation to map from the space of images to the compact target feature space for simple comparison.

C. Indexing Strings

Once a Siamese neural network is trained to convert strings to a feature vector, we must select many process names (or domain names) that we are interested in monitoring (i.e., which names do we expect to be targeted in a spoof attack?). This list is tractable as it is less likely for an attacker to spoof a process (or domain) name that is known by very few people. However, this list can easily grow into the hundreds of thousands. For example, someone interested in monitoring domain names may want to monitor the top 250K common domains around the world. A naive approach is to compute the Euclidean distance between a suspect string's feature vector and every string's feature vector that is being monitored. This brute force nearest search can be improved significantly using indexing.

We employ (randomized) KD-Trees as a geometrical index [3] to quickly search for similar feature vectors. There are several algorithms for performing nearest neighbor search [12], [10], [24], [21], and many may work for this technique. KD-Trees were chosen for their simplicity and availability of open source tools.

In KD-Trees, the dataset is bisected at the median point along the dimension of highest variance, forming two geometric axis-aligned child regions, which are subsequently

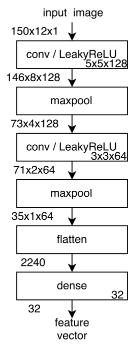


Figure 1. Neural network trained to produce similarity features from image-rendered string queries

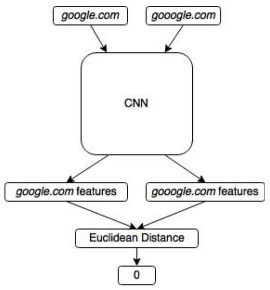


Figure 2. Overview example of training the Siamese Neural network. *google.com* and *gooogle.com* are spoofing pairs and the CNN is trained such that the euclidean distance between their respective features is 0.0.

split using the same logic, and so on, to form a deterministic tree. For search, deterministic trees may scale poorly with dimensionality. Several randomization techniques may be applied to the former strategy, which results in a non-deterministic tree. We use a standard implementation of FLANN [19], in which the split point at each level is chosen randomly among those dimensions that exhibit the greatest variance. A constant number of trees (we use 10 trees in experiments) are built using independent random choices of the split dimension, and all trees are searched for each query.

III. Method

We utilize a Siamese network as a key component for predicting the visual similarity between a query string and a whitelist of potential strings that an attacker may spoof. Our process includes the following steps for determining whether a query is a possible domain or process name spoof.

1) A query string is rendered as a binary image to capture its visual representation. Independent of the query, whitelist strings are rendered into images of fixed size using a common font.
2) From the rendered image, image features are extracted using a neural network, shown in Fig. 1. This network was trained in a Siamese architecture to capture visual similarity between image-rendered strings and possible spoofs. The resulting features are those learned by the Siamese network to best capture image similarity between rendered strings and synthesized spoofs.
3) We query a randomized KD-Tree index for feature vectors with Euclidean distance below a specified threshold to the query feature, and report strings corresponding that correspond to spoofs.

In what follows, we provide additional details about components of this process.

A. Neural network similarity model

The neural network in Fig. 1 is intended to produce a feature vector from an input image of rendered text. In our model, we render images of size 150x12 with white text on black background using `Arial` TrueType font. In our experiments, the image size accommodates horizontal space for 25 characters—an artificial limitation that is trivially extended without other dependent changes in the process.

With well-structured input, our network can be relatively small. We choose two convolution layers with leaky ReLU activations [17], each followed by maxpooling with downsampling. The convolutional layers are followed by a single dense layer that maps the flattened output of the convolutional layers to a 32-dimensional feature vector.

Training the network is using a Siamese architecture in the normal way: a pair of input images $(\mathbf{x}_1, \mathbf{x}_2)$ is compared via Euclidean distance in (1) as $d_{\mathbf{W}}(\mathbf{x}_1, \mathbf{x}_2)$, and are penalized via contrastive loss in (2). Parameters of the network are updated via backpropagation. In our experiments, we use the RMSProp optimizer on batches of 8 images. An example of the entire Siamese CNN is given in Figure 2.

B. KD-Tree Index

Potential targets of spoofing attacks are converted to features vectors with the CNN described above. These feature vectors are indexed using ten randomized KD-Trees, where each tree is grown to purity (1 sample per leaf node). We perform 128 checks on each query unless otherwise specified. The KD-Tree implementation in [20] is used for experiments in this paper.

IV. Results

All experiments are run on two datasets. The first dataset is constructed using the National Software Reference Library (NSRL) [1] using all files with the *.exe* and *.dll* and a filename of at least four characters (not including the extension. Benign pairs (i.e., not spoofing attacks) are created by calculating an all-to-all edit distance and retaining all pairs such that:

$$d(x_1, x_2) \leq 3, \tag{3}$$

where d is the edit distance function (Levenshtein distance). The edit distance of three is fairly small and chosen to make the dataset one that distinguishes visual similarity from edit distance similarity. This data sets helps highlight the shortcomings of various algorithms. Malicious pairs (i.e., spoofing attacks) are created by generating spoofing attacks using the file names extracted from NSRL. Spoofing attacks are generated using thousands of character swaps using both ASCII and unicode characters. The second data set is composed similar to the NSRL data set except that it was generated using 100K active web domains. The restriction on edit distance ($d \leq 3$) was removed when generating the domain data set. This was due to a lack of non-spoofing pairs with distance less than four.

Note that benign strings in both data sets are predominantly composed of ASCII characters. However, this would not be the case when deploying the system in many non-English speaking countries. For this reason, any work using this dataset should not use the presence of unicode as an indicator of spoofing attacks.

A. Setup

For both data sets, we randomly partition the data into training, testing and validation sets. A separate neural network was trained for each data set. The validation set is used during training to prevent over-fitting. Efficacy results are calculated using *Area Under the Curve (AUC)* of the *Receiver Operating Characteristic (ROC)*.

For comparison, the Siamese neural network is compared to two string matching techniques: conventional edit distance and visual edit distance [15], [4].

B. Distance Measure Effectiveness

The first set of experiments compare the effectiveness of the proposed technique to that of conventional edit distance and visual edit distance [15], [4], which we re-implement from descriptions for comparison. Figure 3 shows the ROC for the process name data set. Standard edit distance has an ROC very close to 0.5 (chance). This is expected as all non-spoofing and spoofing pairs had an edit distance not exceeding 3, making it difficult to do significantly better than chance using edit distance alone. Surprisingly, visual edit distance is only slightly improved over edit distance. Lack of improvements highlights the difficulty of manually curating distance measures. The number of possible characters is extremely large when including unicode and manually deriving all to all distances from each character is unfeasible. One could attempt to learn an all to all distance between characters, but manually creating such a data set to learn on is also prohibitively expensive.

Figure 4 shows the ROC curve for the domain data set. Note that all three methods perform far better than the process name data set due to non-spoofing pairs having edit distances that are greater than 3. However, the CNN performs significantly better than the other two techniques. As expected, the visual edit distance is improved over the standard edit distance.

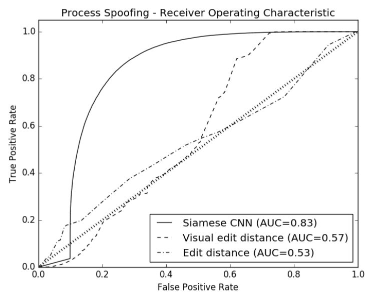


Figure 3. ROC curves for classifying process name spoof attacks

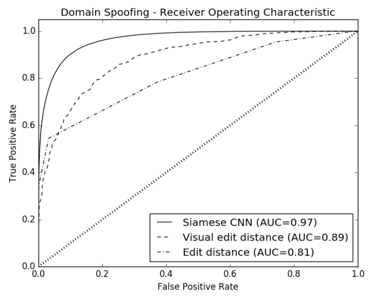


Figure 4. ROC curves for classifying domain name spoof attacks

C. KDTree Performance

The second set of experiments measures the speed improvements and recall degradation when using a KDTrees to index features derived from our model. The KDTree is used to index known strings that may be spoofed. For example, the top 100K most visited domains can be converted to feature vectors using the model and indexed as possible targets for homoglyph attacks. When a new domain is seen, it is converted to a feature vector using the model and is compared to everything in our index. A naive linear scan will take nd computations where n is the number of elements in our index and d is the number of dimensions. On the other hand, a KDTree index will only take $c \times (\log(n) + d)$ where c is the number of checks used by the KDTree. (The number of checks is the number of leaf nodes visited in the search.) We use $c = 128$, and in practice c is typically on the order of 64 to 256 making a KDTree far faster than a naive linear scan for large data sizes. However, this speed increase comes at a cost of lower recall.

Figure 5 and Figure 6 displays the tradeoff between speed and recall with increasing number of checks for the process data and domain data respectively. This experiment is run on 50,000 indexed elements and 50,000 queries. The number of checks equates to the number of leaves explored in a search for the nearest neighbor. The closest element in each explored leaf is returned as the nearest neighbor. The likelihood of finding the true nearest neighbor increases with the number of leaves explored. However, the time it takes

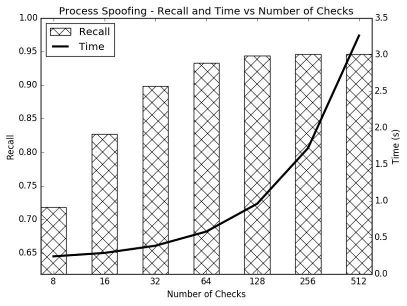


Figure 5. Displays the tradeoff of speed and recall with varying number of checks.

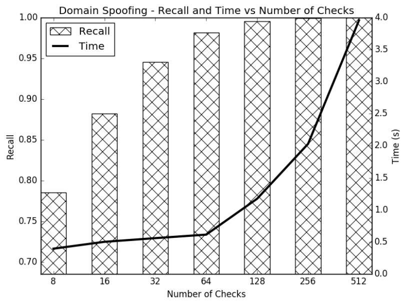


Figure 6. Displays the tradeoff of speed and recall with varying number of checks.

to search also increases.

There is one main differences between the performance of the two data sets. The domain data set achieves near 1.0 recall while the process data set achieves near 0.95 recall. One cause of degradation in the process name data set are clusters of very similar process names. For example, some files in the NSRL data set have versioning information in their file names (e.g., *firefox-1.5.0.1.tar* and *firefox-2.0.0.1.tar*). Each element in these clusters will have very similar feature vectors generated by the CNN making it more likely for a KD-Tree to return incorrect results. Figure 7 shows the distribution of distances from each process/domain name to its nearest neighbor. Distances are calculated using the edit distance normalized by the string length.

As can be seen in Figure 7, process names have a much larger percentage of nearest neighbors falling in the sub 10% range than the domain dataset. This distribution of data can degrade performance as seen in Figure 5.

Both datasets produce nearly identical runtime behavior, and get the best recall/time trade-off with 128 checks. The number of checks was based on 50,000 elements and is expected to increase with the number of elements in the index.

D. Visualizing nearest neighbors

Figure 8 displays the feature vectors of twenty domain names, consisting of 4 domain names each with 4 additional homoglyphs. A PCA projection is performed on these feature vectors to reduce the number of dimensions to two. The names consist of *google.com*, *facebook.com*, *twitter.com*, and *snapchat.com* along with four homoglyph attacks for each domain. Note how each domain and respective homoglyph attacks cluster tightly demonstrating that our learned feature vectors are able to distinguish well between domain names. Distinguishability allows us to predict spoofing attacks with very low false positive rates.

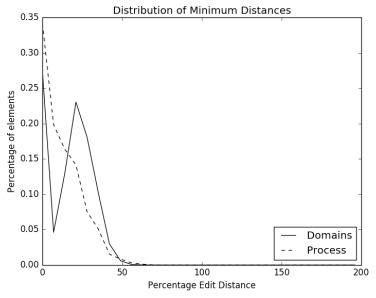
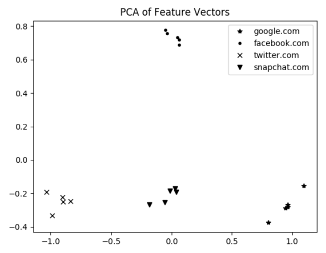


Figure 7. Displays the distribution of distances from each process/domain name to its nearest neighbor. Distance is defined as the percentage edit distance (i.e., the edit distance normalized by the string length).



google.com	facebook.com	twitter.com	snapchat.com
gooogle.com	facebo-öks.com	ţwitter.com	snapclat.com
go0gle.com	tacebook.com	tvvitter.com	snaþchaţ.com
gòoĝle.com	facëbook.com	twitţuŕ.com	snapcąt.com
gøogle.com	facebŏ-oķ.com	ţwiţter.com	sņapchat.com

Figure 8. (top) Two dimensional PCA projection of feature vectors derived from *google.com*, *facebook.com*, *twitter.com*, and *snapchat.com* along with 4 homoglyph attacks; (bottom) homoglyphs for each of the domain names.

V. Conclusion

We presented a technique[2] for detecting domain and process homoglyph attacks using a Siamese CNN. Names are converted to images and passed to the CNN to convert the name to a feature vector. The CNN is trained such that similar strings (i.e., spoofing attacks) generate feature vectors that have a small Euclidean distance while dissimilar strings produce feature vectors that have a large Euclidean distance. Results were compared to conventional detection methods using edit distance and demonstrated a 13% to 45% improvement in terms of area under the ROC curve.

[2]Code and data are publicly available at https://github.com/endgameinc/homoglyph.

References

[1] T. Allen, "National software reference library (NSRL)," 2016.

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

[3] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[4] P. E. Black. (2008) Compute visual similarity of top-level domains. [Online]. Available: https://hissa.nist.gov/~black/GTLD/

[5] J. Bromley, J. W. Bentz, L. Bottou, I. Guyon, Y. LeCun, C. Moore, E. Säckinger, and R. Shah, "Signature verification using a "siamese" time delay neural network," *IJPRAI*, vol. 7, no. 4, pp. 669–688, 1993.

[6] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1. IEEE, 2005, pp. 539–546.

[7] D. Deng, G. Li, and J. Feng, "A pivotal prefix based filtering algorithm for string similarity search," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 2014, pp. 673–684.

[8] D. Deng, G. Li, J. Feng, and W.-S. Li, "Top-k string similarity search with edit-distance constraints," in *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*. IEEE, 2013, pp. 925–936.

[9] S. Garera, N. Provos, M. Chew, and A. D. Rubin, "A framework for detection and measurement of phishing attacks," in *Proceedings of the 2007 ACM workshop on Recurring malcode*. ACM, 2007, pp. 1–8.

[10] T. Ge, K. He, Q. Ke, and J. Sun, "Optimized product quantization for approximate nearest neighbor search," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 2946–2953.

[11] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping," in *Computer vision and pattern recognition, 2006 IEEE computer society conference on*, vol. 2. IEEE, 2006, pp. 1735–1742.

[12] H. Jegou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE transactions on pattern analysis and machine intelligence*, vol. 33, no. 1, pp. 117–128, 2011.

[13] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[14] Y. LeCun, F. J. Huang, and L. Bottou, "Learning methods for generic object recognition with invariance to pose and lighting," in *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, vol. 2. IEEE, 2004, pp. II–104.

[15] A. Linari, F. Mitchell, D. Duce, and S. Morris, "Typo-squatting: The curse"of popularity," 2009.

[16] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Identifying suspicious urls: an application of large-scale online learning," in *Proceedings of the 26th annual international conference on machine learning*. ACM, 2009, pp. 681–688.

[17] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier non-linearities improve neural network acoustic models," in *Proc. ICML*, vol. 30, no. 1, 2013.

[18] S. Marchal, K. Saari, N. Singh, and N. Asokan, "Know your phish: Novel techniques for detecting phishing sites and their targets," in *Distributed Computing Systems (ICDCS), 2016 IEEE 36th International Conference on*. IEEE, 2016, pp. 323–333.

[19] M. Muja and D. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration." *VISAPP (1)*, vol. 2, no. 331-340, p. 2, 2009.

[20] ——, "Flann-fast library for approximate nearest neighbors user manual," *Computer Science Department, University of British Columbia, Vancouver, BC, Canada*, 2009.

[21] ——, "Scalable nearest neighbor algorithms for high dimensional data," vol. 36, no. 11. IEEE, 2014, pp. 2227–2240.

[22] T. R. Trabasso, J. P. Sabatini, D. W. Massaro, and R. Calfee, *From orthography to pedagogy: Essays in honor of Richard L. Venezky*. Psychology Press, 2014.

[23] J. Wang, G. Li, and J. Fe, "Fast-join: An efficient method for fuzzy token matching based string similarity join," in *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*. IEEE, 2011, pp. 458–469.

[24] J. Wang, H. T. Shen, J. Song, and J. Ji, "Hashing for similarity search: A survey," *arXiv preprint arXiv:1408.2927*, 2014.

[25] 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 (TISSEC)*, vol. 14, no. 2, p. 21, 2011.

[26] Y. Zhang, J. I. Hong, and L. F. Cranor, "Cantina: a content-based approach to detecting phishing web sites," in *Proceedings of the 16th international conference on World Wide Web*. ACM, 2007, pp. 639–648.