SEADer: A Social Engineering Attack Detection method based on Natural Language Processing and Artificial Neural Networks

Merton Lansley, Nikolaos Polatidis and Stelios Kapetanakis

School of Computing, Engineering and Mathematics, University of Brighton, BN2 4GJ, Brighton, U.K
M.Lansley@Brighton.ac.uk
N.Polatidis@Brighton.ac.uk
S.Kapetanakis@Brighton.ac.uk

Abstract. Social engineering attacks are one of the most well-known and easiest to apply attacks in the cybersecurity domain. Research has shown that the majority of attacks against computer systems was based on the use of social engineering methods. Considering the importance of emerging fields such as machine learning and cybersecurity we have developed a method that detects social engineering attacks that is based on natural language processing and artificial neural networks. This method can be applied in offline texts or online environments and flag a conversation as a social engineering attack or not. Initially, the conversation text is parsed and checked for grammatical errors using natural language processing techniques and then an artificial neural network is used to classify possible attacks. The proposed method has been evaluated using a real dataset and a semi-synthetic dataset with very high accuracy results. Furthermore, alternative classification methods have been used for comparisons in both datasets.

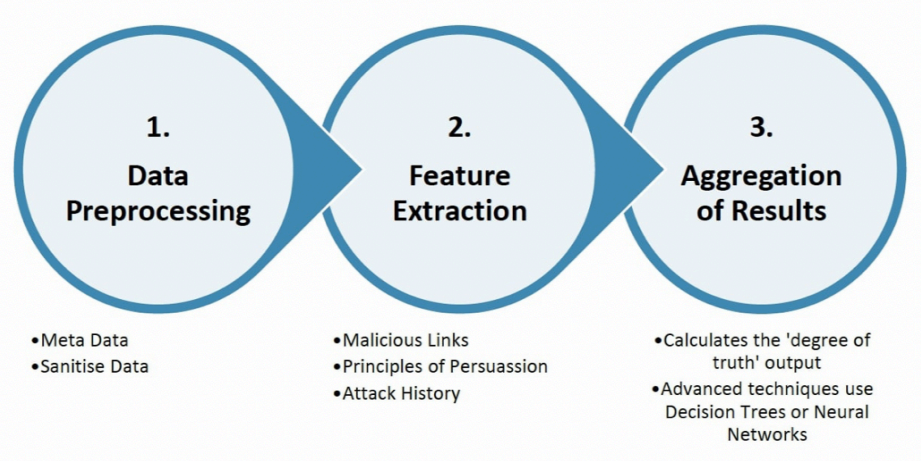
Keywords: Social Engineering, Attack Detection, Online Chat Environments, Natural Language Processing, Neural Networks, Cybersecurity.

1 Introduction

There have been various research projects demonstrating the need for an automated system to recognize SE attacks. Many of these projects show proof of concept models using a variety of different techniques, most commonly Natural Language Processing (NLP) and Machine Learning (ML) methods like Artificial Neural Networks. Whilst these exist, there is a lack of evidence that the theory has been implemented and proved working, with very little evidence of the rate of success [1-4].

Human Psychology plays a large part in the creation of NLP software as it is important to understand how people socialize and behave with others. A common and well-regarded psychologist, Dr. Robert Cialdini proposed the 6 principles of persuasion [5]: Authority, Scarcity, Liking/Similarity, Reciprocation, Social Proof and Commitment/Consistency. Further studies by psychologists also follow a similar consensus that

these are the key factors in defining a persuasive person. Using these key features as a base, Computer Scientists theorized and attempted to build systems that recognized these traits. The research projects on this area can be loosely broken down into three stages, Data Pre-Processing, Feature Extraction and Aggregation of Results as shown in figure 1.



Most research approaches into this problem domain have some relevant data and these data are usually pre-processed. The method of pre-processing differs per system, but generally involves the same goal; preparing the data for classification. The contextual, or meta data, such as time, date and IP addresses are captured in a relationship with the original data. This can be used for correlating different dialogues with one another, one by matching the IP as an internal or external agent and secondly by highlighting potential reoccurring adversaries that may persist under different aliases. The dialogues are sanitized to remove any erroneous content such as HTML tags or corrupt texts. The data can then be parsed using a Natural Language Processing (NLP) parser such as the Stanford CoreNLP [6]. NLP parsers provide a vast array of tools that are able to dissect the human language into dependency trees, define grammatical relations and their Parts of Speech (POS) to name a few common uses. Using these libraries, it allows programmers to create algorithms which are able to classify conversations based on the linguistic features the parser extracts.

This paper is based on the concept that a dialogue is taking place in an online chat environment. To determine if a dialog between two interlocutors is a Social Engineering attack, certain criteria, or else known as features, need to be chosen, a process known as feature selection. The features are selected before the three stages and are commonly based on the principles of persuasion, common phishing tactics like malicious links and the history of the attacker. We can then classify the data based on these features and produce a ‘score’ of how strongly the selected dialog matches the criteria. Each implementation designed to detect each feature will use a variety of classification techniques, such as: Fuzzy Logic, Topic Blacklists, Decision Trees, Random Forest and Neural Networks depending on what needs to be extracted. Furthermore, an automated

system requires the output of a clear decision, albeit a probabilistic decision, this can be done by aggregating the outputs from the Feature Extraction process. The results of each feature are weighted by importance, and at basic level can be averaged to give a Fuzzy logic prediction. By using more advanced techniques such as decision trees or neural networks, the weight of each feature can be calculated programmatically to determine which features carry the most importance regarding whether or not a social engineering attack is taking place.

The following contributions are delivered:

1. A method for detecting social engineering attack in online chat environments is proposed.
2. The proposed method has been evaluated using a real and a semi-synthetic dataset with the results validating our approach.

The rest of the paper is organized as follows: section 2 presents the related work, section 3 delivers the proposed method, section 4 contains the experimental evaluation and section 5 is the conclusions part.

2 Related work

An early implementation of SE detection in real-time telephone systems is SEDA (Social Engineering Defense Architecture) [7]. The researchers mainly focused on identifying repeat callers using their voice signatures. Later a proof of concept model of SEDA [8] was produced being able to correctly identify all attacks in their dataset. A method of detection used by [1] is to identify the questions requesting private information and commands requesting that the user perform tasks they are not authorized to perform. This technique uses a manually derived topic blacklist of verb-noun pairs for which they state should be built around security policies associated with a system. This work is taken further in [2] by identifying 4 main attack; the urgency of the dialog, negative commands and questions, whether the message is likely automated identified by a generic greeting. Finally, they use a reputable cyber security service, Netcraft, to check the safety of a URL. Instead of manual blacklist creation, they use a large corpus of phishing emails to generate a topic blacklist using a Naive Bayes classier. Some approaches like SEADMv2 [3] and MPMPA [9] use complex state machines in order to map out the pathways that can be followed as a checklist-type system to mitigate an attack. This proposal is suited where there are multiple authorization layers that might prevent a request from being carried out. These two separate machines provide some overlap, but the explicit state definition required could be limiting to where these can be applied. The nature of SE attacks is unpredictable meaning that new methods of attacks are always being introduced. The SEADM versions state machines still rely on the user input for changing state, which means the chance for user error and naivety is still present. The works of [4] covers an extensive overview of the existing systems and provide a comprehensive recognition of subsystems for their detection architecture; in

influence, deception, personality, speech act and past experiences. The work of [10] provides a semantic based approach of dialogues to detect social engineering attacks. In [11] the authors show that the human factor is the weakest link in social engineering attacks and based on a human study the prove that. In [12] and [13] the authors provide a theoretical foundation that potentially could be used in real systems to detect attacks., In the works of [14] it is explained how social engineering attacks can potentially be detected, whereas in [15] and [16] different examples of attacks with scenarios are presented. In addition to the works mentioned, there are numerous other articles from relevant domains that can be useful such as the one in [17] where the authors performed an influence analysis of the number of members on the quality of knowledge in a collective, the work in [18] where a neural network is used along with harmony for searching, the one in [19] where nature inspired optimization algorithms for fuzzy control servo systems are discussed and the one found in [20] where an Island-based cuckoo search algorithm with highly disruptive polynomial mutation is proposed.

3 Proposed method

The proposed method consists of several steps to preprocess the dialogs into a dataset for classification. The last steps are applying the classifier. All the steps explained below, have been written in the Python programming language. The SymSpellpy library (a Python port of SymSpell) was used for spelling, the Web of Trust (WOT) Application Programming Interface (API) was used to check any links and finally the SciKit library for the MLP classifier.

In more particular, steps 1 to 5 check for malicious links, steps 6 and 7 determine the spelling quality, and steps 8 to 11 determine the intent of the text, using a predefined blacklist. The score of each feature can be given by: Link score, S_L, Spelling score, S_{sp}, and Intent score, S_I. Each score is then scaled down to between 0 and 1. After the pre-processing of the dialogues (steps 1 – 11), the classification dataset has the following 4 labels: (1) Intent, (2) Spelling, (3) Link and (4) attack or no attack. The final steps use these as inputs for the MLP classifier.

The steps are as follows:

1. Extract all URLs from the dialog text using a regex pattern matcher.
2. If the text contains URLs, send the link/s to the WOT API to evaluate if the web link is malicious.
3. The WOT API returns the reputation of the site (value between 0-100), the confidence of the given reputation (value between 0-100) and the identifying categories (17 in total) that identify the nature of the website. The broad categories and example subcategories are as follows:
    - 1XX Negative (101 Malware, 103 Phishing, 104 Scam, 105 Potentially illegal etc.)
    - 2XX Questionable (201 Misleading claims or unethical, 205 spam, 207 ads / popups etc.)

- 3XX Neutral (301 Online tracking, 302 controversial, 303 political etc.)
- 5XX Positive (501 A good site)

4. If the returned category is of group 1XX or 2XX, then $S_L = 1$.
5. Otherwise, divide the reputation by 100 and take it away from 1 as shown in equation 1.

$$S_L = 1 - \frac{reputation\ value}{100} \tag{1}$$

6. Check for spelling using the SymSpellpy library.
7. Using the best suggested spelling correction, determine the number of misspelled words, given by x. This number is then scaled between 0 and 1 by applying Equation 2. The value of S_{SP} represents the spelling quality, where higher values represents poorer spelling. Rather than a linear function, an exponential function is used to rate a higher number of spelling mistakes more severely. To adjust the rate at which the score tends towards 1, the constant a can be varied to affect how harsh you want to be on spelling errors. After extensive testing it was identified that 0.5 allowed the text to contain a small number of mistakes without creating a high score. For example, if a is set to 0.5 and $x = 1$ then S_{SP} = 0.39, if $x = 5$ then $S_{SP} = 0.92$.

$$S_{SP} = 1 - e^{-ax} \tag{2}$$

8. The next step uses the corrected spelling of the dialog, and checks it against a blacklist, derived from 48 security policy style words. This can be easily populated with company or environment related words such as: credentials, passwords, database etc. The number of blacklist matched words is given by M_B.
9. The algorithm at this step checks for intent verbs and adjectives such as need, must, urgent etc. This value is given by M_I
10. To tune the results, values M_B and M_I are multiplied by the weights W_B and W_I, weighted at values of 2 and 1 retrospectively. This step has been added as an equation in case someone wants to change the weight of this step, if it is considered more important. The value x can hence be given by equation 3.

$$x = (M_B \times W_B) + (M_I \times W_I) \tag{3}$$

11. Then, the value of x is normalized in equation 4, using the same exponential function as equation 2. Where $a = 0.4$ to give the best output. A higher value of S_I indicates a higher concentration of blacklisted words in the text.

$$S_I = 1 - e^{-ax} \tag{4}$$

12. At this step the original dialogue dataset is being checked to identify which dialogue was indeed an attack and assign the true (1) or false (0) value to the new dataset used for classification.

13. The dataset is populated and the MLP classifier is applied. If the output is high, then it is considered an attack otherwise it is not considered an attack. Initially, we define an activation function as: $g(z)$ with x input values and w weights as input. The activation function is shown in equation 7.

$$z = w_1x_1 + w_2x_2 + \cdots + w_mx_m \tag{7}$$

If g(z) is greater than a given threshold theta, the output is 1 or -1 otherwise, a s shown in equation 8.

$$g(z) = \begin{cases} 1 \text{ if } z > \text{theta} \\ -1 \text{ otherwise} \end{cases} \tag{8}$$

Where $z = w_1x_1 + w_2x_2 + \cdots + w_mx_m = \sum_{j=1}^{m} x_jw_j = w^Tx$

At the next step, we use Rosenblatt's perceptron rule to update the weights as follows: (a) Each weight was initialized with small random numbers and (b) for each iterative training step for each input x the output value was calculated, and the weights were updated.

The output update is defined in equation 9, with $\Delta w_j = e\left(target(i) - output(i)\right)x_j^i$ and e is the learning rate, *target* the actual (true) class label and *output* the predicted label. Weight updates were iterative, and updates were performed simultaneously.

$$w_j^+ = w_j + \Delta w_j \tag{9}$$

4 Experimental evaluation

The experimental evaluation took place in an offline environment using a real dataset and a semi-synthetic dataset, which we call the standard dataset and the compound dataset. Following from section 3, we have built both datasets using a social engineering attack dataset with 147 entries classified as an attack or not. After the preprocessing steps followed in section three, we derived the following four classification labels: (a) intent (b) spelling (c) link (d) is attack[1]. Furthermore, the standard dataset contains 147 entries obtained from [12], while the compound dataset is based on the 147 entries plus 600 entries from customer support-based tweets from Twitter, none of which are classified as attacks. Both dataset text entries have been converted to numerical based classification datasets with four entries each. Three labels for the respective data and one with a yes or no (1 or 0) value of a conversation being a social engineering attack or not. To both datasets a small number of links to websites have been added, with some being malicious.

The accuracy metric has been used for the evaluating the classification models. The metric calculates the fraction of the prediction that each model got right. Equation 10 represents the accuracy where TP stands for true positives, TN for true negatives, FP for false positives and FN for false negatives.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{10}$$

The results of the evaluation as shown in tables 1, 2, 3, 4, 5, 6, and 7 are based on the standard and compound datasets respectively and a 5-fold cross-validation approach has been used. In tables 1 to 6 the MAX and MIN values represent the maximum and minimum values of the 5-fold returned after each time the algorithm ran and the MEAN is the median value. Each algorithm was executed 10 times and at the end of each table the value is the mean value of 10 iterations for each of the three table labels. Table 7 however, presents the average mean results for the standard and compound datasets respectively.

Three algorithms from the SciKit machine learning library have been used for comparison purposes: Decision Tree, Random Forest and Neural Network multi-layer perceptron. The settings used for the algorithms were the following:

Decision Tree: SciKit learn settings with default settings which included unlimited leaf nodes.

Random Forest: SciKit Random Forest classifier with 50 Estimators.

Neural Network: SciKit MLP classifier with lbfgs solver, alpha = 1e-5 and hidden layer size = (15,15,15).

[1] The datasets can be found at https://github.com/npolatidis/seader

Table 1. Decision Tree results for the standard dataset

MEAN	MAX	MIN
0.736231884	0.826086957	0.583333333
0.692885375	0.782608696	0.625
0.68442029	0.833333333	0.458333333
0.650362319	0.695652174	0.583333333
0.656347826	0.76	0.608695652
0.659914361	0.826086957	0.5
0.694762846	0.826086957	0.416666667
0.647463768	0.791666667	0.5
0.65	0.695652174	0.583333333
0.741798419	0.833333333	0.652173913
0.681	**0.787**	**0.551**

Table 2. Random Forest results for the standard dataset

MEAN	MAX	MIN
0.683333333	0.791666667	0.608695652
0.631521739	0.708333333	0.565217391
0.691304348	0.791666667	0.608695652
0.657608696	0.75	0.565217391
0.651811594	0.782608696	0.333333333
0.700724638	0.75	0.666666667
0.647826087	0.75	0.5
0.736067194	0.782608696	0.708333333
0.699130435	0.8	0.608695652
0.72826087	0.826086957	0.583333333
0.683	**0.773**	**0.575**

Table 3. Multi-Layer Perceptron results for the standard dataset

MEAN	MAX	MIN
0.709057971	0.75	0.652173913
0.74384058	0.826086957	0.652173913
0.673043478	0.8	0.565217391
0.640217391	0.782608696	0.47826087
0.75273386	0.833333333	0.625
0.65085639	0.739130435	0.541666667
0.751449275	0.826086957	0.565217391
0.659826087	0.739130435	0.56
0.68173913	0.8	0.52173913
0.65	0.695652174	0.583333333
0.691	**0.779**	**0.574**

Table 4. Decision Tree results for the compound dataset

MEAN	MAX	MIN
0.908050847	0.916666667	0.9
0.921481816	0.957983193	0.900826446
0.921386555	0.941176471	0.890756303
0.923023127	0.95	0.890756303
0.921470588	0.957983193	0.9
0.923120703	0.941176471	0.909090909
0.909774011	0.93220339	0.891666667
0.919747899	0.932773109	0.908333333
0.913009121	0.925619835	0.899159664
0.914705882	0.933333333	0.890756303
0.918	**0.939**	**0.898**

Table 5. Random Forest results for the compound dataset

MEAN	MAX	MIN
0.902910619	0.95	0.857142857
0.919787138	0.941176471	0.900826446
0.909703336	0.924369748	0.899159664
0.92640056	0.941666667	0.915966387
0.921414566	0.95	0.883333333
0.919803922	0.941176471	0.891666667
0.916468927	0.940677966	0.883333333
0.911358543	0.933333333	0.883333333
0.931468273	0.941666667	0.917355372
0.911372549	0.916666667	0.907563025
0.917	**0.938**	**0.894**

Table 6. Multi-Layer Perceptron results for the compound dataset

MEAN	MAX	MIN
0.929803922	0.949579832	0.908333333
0.921442577	0.932773109	0.908333333
0.926414566	0.941666667	0.915966387
0.923107345	0.933333333	0.908333333
0.936442577	0.949579832	0.915966387
0.919789916	0.949579832	0.891666667
0.923120934	0.949579832	0.899159664
0.924733894	0.941176471	0.899159664
0.924747899	0.941666667	0.907563025
0.921398477	0.949579832	0.907563025
0.925	**0.944**	**0.906**

Table. 7. Comparison of algorithms for the standard and the compound datasets

(a)

ALGORITHM	RESULT
Decision Tree	0.681
Random Forest	0.683
Multi-Layer Perceptron	**0.691**

(b)

ALGORITHM	RESULT
Decision Tree	0.918
Random Forest	0.917
Multi-Layer Perceptron	**0.925**

5 Conclusions

In this paper we presented a novel method for social engineering attack detection based on natural language processing and artificial neural networks. The method initially processes a dialogue and then creates a dataset that can be used for classification purposes. The proposed method has been evaluated using a real and a semi-synthetic dataset and can detect social engineering attacks with very high accuracy. Furthermore, alternative classification methods have been used for comparison in order to show the effectiveness of our method.

Although the accuracy results are high, we believe that there is still room for improvement, thus in the future we plan investigate the possibility of adding more features to the dataset and to apply a deep neural network for the classification process. In addition to that we aim to investigation performance and optimization issues.

References

1. Sawa, Yuki, Ram Bhakta, Ian G. Harris, and Christopher Hadnagy. "Detection of social engineering attacks through natural language processing of conversations." In *2016 IEEE Tenth International Conference on Semantic Computing (ICSC)*, pp. 262-265. IEEE, 2016.
2. Peng, Tianrui, Ian Harris, and Yuki Sawa. "Detecting phishing attacks using natural language processing and machine learning." *2018 IEEE 12th International Conference on Semantic Computing (ICSC)*. IEEE, 2018.
3. Mouton, Francois, Alastair Nottingham, Louise Leenen, and H. S. Venter. "Finite state machine for the social engineering attack detection model: SEADM." *SAIEE Africa Research Journal* 109, no. 2 (2018): 133-148.
4. Tsinganos, Nikolaos, Georgios Sakellariou, Panagiotis Fouliras, and Ioannis Mavridis. "Towards an Automated Recognition System for Chat-based Social Engineering Attacks in Enterprise Environments." In *Proceedings of the 13th International Conference on Availability, Reliability and Security*, p. 53. ACM, 2018.
5. Cialdini, Robert B. "The science of persuasion." *Scientific American* 284, no. 2 (2001): 76-81.
6. Manning, Christopher, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. "The Stanford CoreNLP natural language processing toolkit." In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pp. 55-60. 2014.

7. Hoeschele, Michael, and Marcus Rogers. "Detecting social engineering." *IFIP International Conference on Digital Forensics*. Springer, Boston, MA, 2005.
8. Hoeschele, Michael. "CERIAS Tech Report 2006-15 DETECTING SOCIAL ENGINEERING." (2006).
9. Jamil, Abid, Kashif Asif, Zikra Ghulam, Muhammad Kashif Nazir, Syed Mudassar Alam, and Rehan Ashraf. "MPMPA: A Mitigation and Prevention Model for Social Engineering Based Phishing attacks on Facebook." In *2018 IEEE International Conference on Big Data (Big Data)*, pp. 5040-5048. IEEE, 2018.
10. Bhakta, Ram, and Ian G. Harris. "Semantic analysis of dialogs to detect social engineering attacks." *Proceedings of the 2015 IEEE 9th International Conference on Semantic Computing (IEEE ICSC 2015)*. IEEE, 2015.
11. Heartfield, Ryan, and George Loukas. "Detecting semantic social engineering attacks with the weakest link: Implementation and empirical evaluation of a human-as-a-security-sensor framework." *Computers & Security* 76 (2018): 101-127.
12. Bezuidenhout, Monique, Francois Mouton, and Hein S. Venter. "Social engineering attack detection model: Seadm." *2010 Information Security for South Africa*. IEEE, 2010.
13. Mouton, Francois, Louise Leenen, and H. S. Venter. "Social engineering attack detection model: Seadmv2." *2015 International Conference on Cyberworlds (CW)*. IEEE, 2015.
14. Nicholson, James, Lynne Coventry, and Pam Briggs. "Can we fight social engineering attacks by social means? Assessing social salience as a means to improve phish detection." *Thirteenth Symposium on Usable Privacy and Security ({SOUPS} 2017)*. 2017.
15. Krombholz, Katharina, Heidelinde Hobel, Markus Huber, and Edgar Weippl. "Advanced social engineering attacks." *Journal of Information Security and applications* 22 (2015): 113-122.
16. Mouton, Francois, Louise Leenen, and Hein S. Venter. "Social engineering attack examples, templates and scenarios." *Computers & Security* 59 (2016): 186-209.
17. Nguyen, Ngoc Thanh, Van Du Nguyen, and Dosam Hwang. "An influence analysis of the number of members on the quality of knowledge in a collective." *Journal of Intelligent & Fuzzy Systems* 32.2 (2017): 1217-1228.
18. Saadat, Javad, Payman Moallem, and Hamidreza Koofigar. "Training echo state neural network using harmony search algorithm." *Int. J. Artif. Intell* 15.1 (2017): 163-179.
19. Precup, Radu-Emil, and Radu-Codrut David. *Nature-Inspired Optimization Algorithms for Fuzzy Controlled Servo Systems*. Butterworth-Heinemann, 2019.
20. *Bilal H. and Abed-alguni. "*Island-based Cuckoo Search with Highly Disruptive Polynomial Mutation". *Int. J. Artif. Intell* 17.1 (2019): 57-82.