RESEARCHERID
THOMSON REUTERS

[Tabassum * *et al.*, 7(4): April, 2018]
IC™ Value: 3.00

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

IJESRT

INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY

BEATING THE ARTIFICIAL CHAOS: FIGHTING OSN SPAM USING ITS OWN UNDERLYING TEMPLATES BY TANGRAM

Tayyaba Tabassum[*1], Lubna Kauser[2] & Shaik Abid[3]
[*1&2]Asst. Professor, KBNCE, Klb-585104
[3]Ph.D., Scholar, SSSTUMS, MP-466001

DOI: 10.5281/zenodo.1228729

ABSTRACT

Internet users are regularly using Online social networks (OSNs). However, spam originating from various sources causes damage to less security-savvy users. Earlier counteraction withstand OSN spam from different angles. Due to the wide range of spam, there is rarely any current procedure that can independently detect the majority of OSN spam. In this paper, we empirically analyze the textual pattern of a large collection of OSN spam. An inspiring finding is that the majority (e.g., 76.4% in 2015) of the collected spam is generated with underlying templates. Based on the analysis, we propose tangram, an OSN spam filtering system that performs online inspection on the stream of user-generated messages. Tangram extracts the templates of spam detected by existing methods and then matching messages against the templates toward the accurate and the fast spam detection. It automatically divides the OSN spam into segments and uses the segments to construct templates to filter future spam. Results on Twitter and Facebook data sets show that tangram is accurate and can rapidly generate templates to throttle newly emerged campaigns. Furthermore, we analyze the behavior of detected OSN spammers. We find a series of spammer properties—such as spamming accounts are created in bursts and a single active organization orchestrates more spam than all other spammers combined—that promise more widespread spam counteraction

Keywords: Online social networks, spam, spam campaigns, tangram.,spam detection.

I. INTRODUCTION

OSN became popular spammers started exploiting online social networks(OSNs). Researchers propose combating OSN spam from different angles, including mining the textual content, studying the redirection chains of embedded URLs, as well as classifying the URL landing pages. Despite the development of countermeasures, spammers find their way to adapt and stick. Back to 2011, on Twitter, one of the most popular OSNs nowadays, more than 5% of collected tweets are spam, which has slipped through all the deployed defense mechanism. While up to 2014, 5% of the entire Twitter's user base are spam bots. Why could this still happen given the efforts to build various spam mitigation systems? We observe that a primary reason is the absence of clear understandings of what techniques the spammers are using to construct OSN spam and how the techniques evolve. Such missing piece of fundamental information is critical for exploring effective designs to throttle OSN spam. Toward uncovering OSN spam generation techniques, we conduct a large-scale, consecutive measurement study. We find that the majority of spam is generated with underlying templates, which is consistent with prior email spam research . Templates are valuable for spammers, because they let spammers control and customize the semantic meaning of generated messages to boost the conversion rate. OSN spammers have evolved to use more sophisticated templates that break the assumptions in prior email spam template generation research, making them ineffective for OSN spam. In particular, our measurement results reveal three challenges that render template-based OSN spam hard to throttle.

Absence of Invariant Substring in Template: Prior spam template generation research made a crucial assumption that an invariant substring is hard-coded in a template, so that every instantiation of the template contains such string. Unfortunately, an OSN spam template does not always contain any invariant substring.

Prevalence of Noise: Spammers extensively add semantically unrelated noise words into spam messages. The presence of noise diversifies spam, and increases the difficulty to identify semantically meaningful text segments.

Spam Heterogeneity: Spam instantiating different templates mixes with spam without any underlying templates. It is hard to obtain a training set with a single template in an online detection scenario. In our project, we propose Tangram system to combat OSN spam through effective spam-template generation. Tangram stands out among existing spam countermeasures because of three properties. First, Tangram directly tackles spam whereas many existing methods detect spammers instead of spam. Such methods are based on account activity and need long observation periods for the account features to accumulate. Second, some other detection approaches are based on URL analysis, which inherently cannot detect spam without URLs. Researchers have revealed significant amount of such spam. The few existing methods that detect spam with or without URLs in real-time suffer from high false positive rates. In contrast, Tangram is the first accurate online OSN spam detection system that detects spam with or without URLs. Third, what is more unique to Tangram is that it directly hits OSN spam's vital point—template-based spam that counts the most. Tangram extracts templates of spam detected by existing methods and then matches messages against the templates toward accurate and fast spam detection. Beyond spam detection, we further investigate the detected spammers to infer their strategy.
In summary, Tangram is highly accurate because of the following sweet spots.

Embrace the Absence of Invariant Substring: We identify frequently appearing segments within messages and then locate equivalent segments among messages. Such segments are later assembled into spam templates for matching future spam.

Mitigate the Prevalence of Noise: We cast a sequence-labeling task to label each word in a given message as either "noise" or "non-noise". Only "non-noise" words yield templates.

Break Spam Heterogeneity: We pre-cluster spam and perform template generation within individual partitions. We also discard outlier messages in the partition.

Build a Double Defense: We mitigate spam without underlying templates using a supplementary module that detects spam with excessive semantically unrelated noise words.

In addition, we can provide Tangram with multiple heterogeneous detection modules in practice.

II. MATERIALS AND METHODS

1. Existing system

Tangram builds template-based spam detection on top of existing detection methods toward higher accuracy and speed. It generates the underlying templates of spam detected by various existing methods. It then uses the templates to accurately, quickly match and detect spam. Figure 1 depicts the Tangram workflow. It takes a stream of raw messages as input, and classifies them as either spam or legitimate online. After the classification, spam is filtered, while legitimate messages pass through. Two components can classify messages: the template matching module and the auxiliary spam filter. The template matching module, along with the template generation technique, is our major contribution. The auxiliary spam filter, on the other hand, supplies training spam messages. It can be any deployed spam filter, e.g., a blacklist spam filter. In our paper, we propose Tangram system to combat OSN spam through effective spam-template generation. Tangram stands out among existing spam countermeasures. In contrast, Tangram is the first accurate online OSN spam detection system that detects spam with or without URLs. Third, what is more unique to Tangram is that it directly hits OSN spam's vital point—template-based spam that counts the most. Tangram extracts templates of spam detected by existing methods and then matches messages against the templates toward accurate and fast spam detection. Beyond spam detection, we further investigate the detected spammers to infer their strategy.

Liabilities:

- No Privacy for Users.
- Storing data is very difficult for users.
- In the present social network application , are not having facility to share or view templates.

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

- In the present social network application prevents spam for images and text.

2. Proposed system

We reuse the same parameters reported. The auxiliary spam filter is not an oracle. It may mistakenly report legitimate messages as spam, or miss to report spam messages. Note that in what follows we evaluate the detection accuracy of only the modules proposed, but not the accuracy of the auxiliary spam filter.

As before mentioned, we focus on the accuracy of our proposed modules—template matching and noise detection. We first try to analytically capture how their accuracy varies with that of auxiliary filter. Let Ts and Tl represent the number of spam tweets and the number of legitimate tweets in the ground truth.

Benefits:

- We can provide much safety for users.
- Users can access account from anywhere.
- In the proposed system social network application are having facility to share or view templates.
- In the proposed system social network application it does not prevents spam for images and text.

3. Tangram: template-based spam detection system

we present Tangram, an accurate and fast template-based spam detection system. We first formulate the notions of template, template matching and template generation. Next, we detail the online Tangram system.

System Design Overview:

Tangram builds template-based spam detection on top of existing detection methods toward higher accuracy and speed. It generates the underlying templates of spam detected by various existing methods. It then uses the templates to accurately, quickly match and detect spam. It takes a stream of raw messages as input, and classifies them as either spam or legitimate online. After the classification, spam is filtered, while legit0imate messages pass through. Two components can classify messages: the template matching module and the auxiliary spam filter. The template matching module, along with the template generation technique, is our major contribution. The auxiliary spam filter, on the other hand, supplies training spam messages. It can be any deployed spam filter, e.g., a blacklist spam filter. Template Matching and Template Generation: We define a template to be a sequence of macros of two types, dictionary and noise .We represent a dictionary macro as a set of values separated by "|" and a noise macro as Thus, templates produced by Tangram are naturally encoded as regular expressions, specifically concatenations of clauses and Template matching matches a given message against the corresponding regular expression. A successful template match implies the tested message instantiates the template, and should be flagged as spam. We define template generation as the task of inferring the template's regular expression representation from a set of observed spam instances. Initially the template matching module is not equipped with any template, so all messages will pass through. However, if a message is blocked by the auxiliary spam filter, it is treated as an instantiation of an unexpected template, and is saved in the spam buffer. Once the number of messages in the spam buffer exceeds a predefined window size threshold, the system invokes the template generation procedure, and deploys the newly generated templates in the template matching module. As spam categorization demonstrates, spam messages are with or without underlying templates. We first pre-process spam messages in the buffer into campaigns. Messages belonging to no campaign are not template-based or its same-template messages are not sufficient enough. We then feed only messages in campaigns to template generation. For the left spam tweets in the buffer, we wait for 10 times of window size until we evict them from the buffer. Whenever a newly generated template happens to be highly similar to an existing one, we will merge their regular expressions. The template generation first identifies the subset of spam messages sharing the same template. These messages are tokenized into sequences of words. After executing noise detection , we are left with spam content generated by dictionary macros. We divide every message into the same number of segments. Each segment, containing zero or more tokens, corresponds to one macro in the template. We then construct the macro by combining the segment's unique strings across messages with the concatenation of the macros for all segments constitutes the complete template. Inferring the number of segments and which tokens belong to which segment are key challenges in template generation.

1. We use the heuristic of preferring more compact templates (i.e., shorter regular expressions) that match all of the input spam messages without using wild cards. This heuristic follows the traditional approach of preferring simpler descriptions to more complex ones; our experiments validate its effectiveness. Furthermore, finding the shortest template for a set of messages is an NP-hard problem. We develop a practical approximation as follows.

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

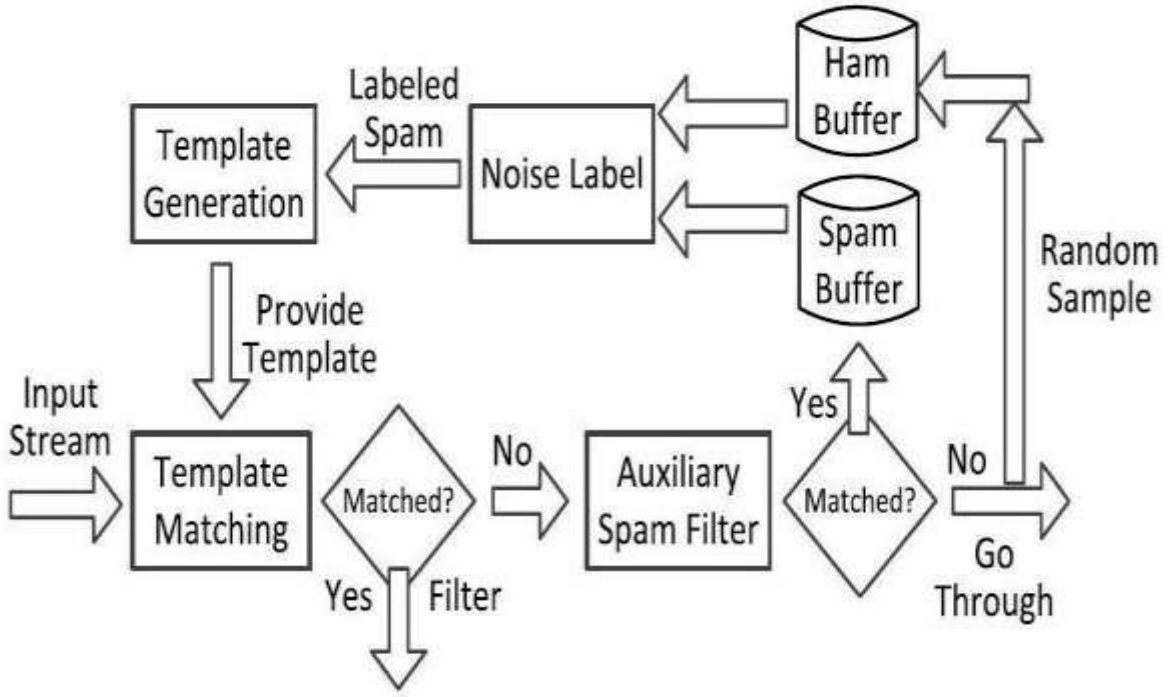

Fig.1. Tangram framework: The template generation and matching overview

4. Single campaign template generation

For ease of presentation, we first introduce the approach to generate a single template given spam instantiating the same underlying template. It is the basis of Tangram. We expand the approach to generate templates on a mixture of spam instantiating multiple templates. A strength of our approach is generating templates without any invariant substring. However, we do expect that some non-trivial subset of a campaign will share a common substring, because the dictionary macro may instantiate to the same textual content when multiple spam messages are generated. This property helps infer the correspondence of segments between messages.

Since substrings shared by subsets of a campaign are crucial for template inference, a naive alternative is to break a campaign apart so that each part contains an invariant substring, then reuse the existing template generation algorithm. Unfortunately, such a immature alternative cannot capture as much spam as our technique does. Using the sample campaign, if we break the campaign into two parts, the first two messages and the last three messages, the templates generated by the naïve alternative cannot capture the unobserved message starting with “RIP Celeb C”, whereas our technique can detect such case.

We systematically exploit such substrings shared by subsets of a campaign in three steps, common super sequence computation, column concatenation, and regular expression representation.

Common Super sequence Computation*:*

The first step is to compute the messages’ common super sequence. Shortest common super sequence is an NP-hard problem .We use an approximation algorithm named Majority-Merge because of its simplicity. It takes n sequences as input and initializes the super sequence, s, as an empty string. It iteratively chooses the majority of the leftmost tokens of the input sequences, denoted as a, and appends a to s. Meanwhile, the leftmost a is deleted from the input sequences. It repeats this step until all sequences are empty and outputs s. For ease of understanding the Majority-Merge algorithm. In step 1, both “Big” and “Celebrity” represent the majority of the leftmost tokens of the input tweets. We, however, assign a with “Big” because its corresponding input tweet precedes the one that “Celebrity” corresponds to. (This rule applies also to subsequent steps .We first delete “Big” from messages prefixed with it (i.e., the first one and the third one). We then use the updated tweet set as the input of step 2. Again, step 2 finds two tokens, “Name” and “Celebrity”, that dominate the majority of the leftmost tokens. We select “Name” as a according to the rule in step

1. After six steps of similar operation, the (intermediate) output super sequence s becomes "Big Name A Celebrity B an. In the final output super sequence, each token is trivially a substring shared by some subsets of the campaign. Desirable substrings are i) shared by large subsets and ii) long.

We achieve goal i) by producing a shorter super sequence via the following two phases: matrix representation and matrix column reduction.

- **Matrix Representation:**

We build a matrix during the execution of Majority-Merge algorithm. The header row is the final super sequence output by the Majority-Merge algorithm. Each of the remaining rows represents one input sequence, that is, one spam tweet. If the ith sequence is picked in step j for extracting token a, the cell at row i, column j will be assigned the token a. (We also call the token a as column label.) Otherwise, the cell will be assigned an empty string. It corresponds to the first step of the Majority-Merge algorithm. Since the chosen token a is "Big", which leads the first and the third tweets, the first row and the third row are labeled with "Big"; Naturally, the concatenation of labels of each row is exactly the row's corresponding input sequence. We denote this property as the super sequence property.

- **Matrix column reduction**:

To produce a shorter super sequence, we need to merge columns that share the same label, while maintaining the super sequence property.

After merging two cells from two columns, the new cell will be assigned the column label if either cell before merging has been assigned so. Take the two columns sharing "offensive". After merging them, the first two new cells will be assigned ε because all corresponding cells are ε. On the other hand, each of the other three new cells will be assigned "offensive", which is the column label of one corresponding cell before merging. These two columns are merged into one column containing "offensive". Without loss of generality, we state the three sufficient conditions that determine whether column k can be merged into column j without affecting the super sequence property Note that the merging is directional, after which column j is kept while column k is deleted. Condition

- column j and column k have identical label; Condition
- in any row at least one column is ε; and Condition
- if the cell at row i, column k is not, all cells in row i, between column j and column k must be Table V shows the column merging result. Noticeably, the repeated columns of "offensive content, look at this video" is gone after the merging, yielding a more compact matrix representation.

- **Column Concatenation:**

To achieve goal ii) for obtaining long substrings shared by subsets of campaigns, we further concatenate the matrix columns obtained from the previous step. Column concatenation also operates on a column pair, after which each cell becomes the concatenation of the two corresponding cells. Different from column merging, column concatenation does not require the target columns to share identical label. It only requires that the value of the corresponding, non-ε cells in the two columns has mapping. For example, the first two columns are concatenated because "Big" always maps to "Name but the fifth and the sixth columns in cannot be concatenated because "B" maps to two values, "an" and The effect of column concatenation is two-fold. First, it moves multiple tokens into one cell, revealing the true template by assembling tokens (words) into word phrases. For example the three separate columns "Big", "Name", and "A" become one celebrity name. Second, the cells on the same column after column concatenation may have different contents, like the first two columns .This maps to the dictionary macro case, where different cell contents are different instantiation of the dictionary macro.

- **Regular Expression Representation:**

Converts the matrix into a regular expression to represent the generated template. We initialize the regular expression representation to be an empty string, s. Then we iterate through each column. If all the cells in the column share an identical value, we append the value to s. Otherwise, we make a "|" clause by concatenating all the unique values with "|", and append the clause to. The header row of gives an example of the generated regular expression representation. Finally, we add a and a to the head and the tail of s to respectively mark the beginning and the ending of a message.

5. Multi campaign template generation:

We now expand single campaign template generation to multi-campaign scenarios over spam instantiating different templates or even without underlying templates. We first separate the spam into distinct campaigns automatically and then individually invoke the single campaign template generation.

We first use single-linkage clustering to group messages that share at least k consecutive identical tokens, k being a system parameter. The goal is to put semantically similar messages in the same cluster, while separating semantically different messages into different clusters. The transitive closure of these links forms our initial clustering. This clustering does not require every message pair in the cluster to share an invariant substring. We use a small training set of collected spam tweets to choose the value of k experimentally. The value of k is not sensitive to the training set size. For example, we test with size 10,000, 5,000 and 2,000 and obtain consistent results. With a training set of size 10,000, a loose threshold (e.g., k = 3) results in a big cluster containing 42% of the spam, while spam messages in this cluster have different semantic meanings like Lady Gaga, Apple product and so on. A tight threshold (e.g., k >= 5) results in a large number of small clusters, where multiple clusters share the same semantic meaning. For example, 9 out of the 20 largest clusters in the experiment should be merged. In comparison, k = 4 produces the best result in our experiments. We suggest that k be empirically adjusted. Given a test dataset, one can first extract a subset of test tweets and run template generation over it with an initial k. If it results a large number of smaller clusters, we need to shrink k. Otherwise, we increase k instead. When it goes to practical filtering of real-time tweet stream, we again track a subset of messages and assign an initial k first. We then post-analyze the detection result of the subset. If many obvious spam tweets are not detected, we need to improve template generation precision by increasing k. We then refine the clusters using the single campaign template generation algorithm. Intuitively, spam messages from different campaigns will result in non-compact templates, a fact we utilize to identify which spam should be removed from a given cluster. We explain this process using the dataset as a running example. Specifically, we find a row to remove if the number of ε is larger than a predefined threshold w × p, where w denotes the word count (except notations and URLs) of the dataset/matrix and p is a systematical parameter. We set p no larger than the reciprocal of average word count per column. The reason is as follows: after column concatenation , we can treat words in each cell as one new larger word. Then w times the reciprocal of average word count per column approximates the number of such new larger words. If ε is more than such approximation, we consider the matrix as non-compact. It contains nine, which is larger than 43 × 0.2 = 8.6 and we need to remove certain rows and all-ε columns to make it more compact. We first find the column with the most that is, the fourth column. We then remove any row corresponding to non-ε words in the fourth column, that is, the last row. After deletion, the fourth column contains only ε, which should be removed as well. We repeat the above process over the new matrix.

6. Noise labeling:

Spam messages often mention other users, popular terms and hash tags unrelated to the semantics of the rest of the messages. Such content helps expose spam to a larger audience, because users may search or browse tweets by topic. It also diversifies spam and makes detection difficult. We refer to this type of content as noise. Popular forms of noise include celebrity names, TV shows, trending hash tags and many others. We next elaborate how noise affects template generation and design a model to automatically label noise given a small amount of easily, manually labeled noise as trained data. Once trained well, the model can accurately label noise tokens in real-time stream of spam tweets during Tangram execution. Noise creates extra difficulties for template generation. If the generated template contains a segment of noise, the template will be too “specific”, in the sense that it cannot match the spam with a different sequence of noise terms. In addition, spam instantiating different templates may coincidentally share an identical sequence of noise terms. It increases the chance to mislead the template generation module so that it attempts to extract a single template for them. Thus, we first perform a pre-processing step to identify noise tokens in the messages, and then effectively ignore them when generating the template (i.e., we replace them with a wildcard that matches anything).

We treat noise detection as a sequence labeling task, in which the goal is to automatically label each token in the tweet as noise or non-noise. We employ a standard sequence-labeling approach, Conditional Random Fields (CRFs). The CRF is a model, learned from training data, that infers a label for each token in a given messsages. The model exploits regularities in the features of noise and non-noise tokens (detailed below), as well as regularities in label sequences. The CRF requires identifying a set of features for each token that are relevant to the task. In our case, we found a set of features that appear to be highly indicative of noise. The key observation is that noise terms are popular, yet unrelated to each other and to other elements of the messages. We would expect regions of noise to contain individual tokens that are common on messages, but transitions between tokens that are relatively uncommon. We capture these intuitions with three numeric features. Let freq(s) represent the frequency of a string s, which we measure of a large set of unlabeled messages. For each token ti in a messages, we create the following three features in the CRF: freq(ti),freq(titi+1)2/(freq(ti)freq(ti+1)), and freq(ti−1ti)2/(freq(ti−1) freq(ti)). The first feature captures the popularity of the token, whereas the second and

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

third estimate how likely ti is to occur given the surrounding tokens. We processed these features into five discrete quintiles for incorporation into the CRF. We further add four orthographic features to capture common elements of noise terms. They indicate whether ti is capitalized, is numeric, is a hashtag, or is a user mention @. To train our CRF, we hand-labeled 1,000 messages as training data, manually identifying each token as noise or non-noise. We then employed this learned model on each messages before template generation. In a separate experiment on the labeled messages, we found that our trained CRF correctly labeled an average of 92% of test-set tokens as noise or non-noise.

Fig1: Registration Phase

Fig2: Registration Details

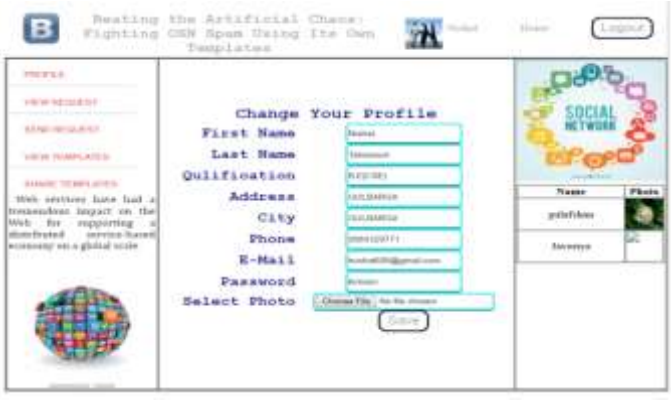

Fig3: Edit Profile

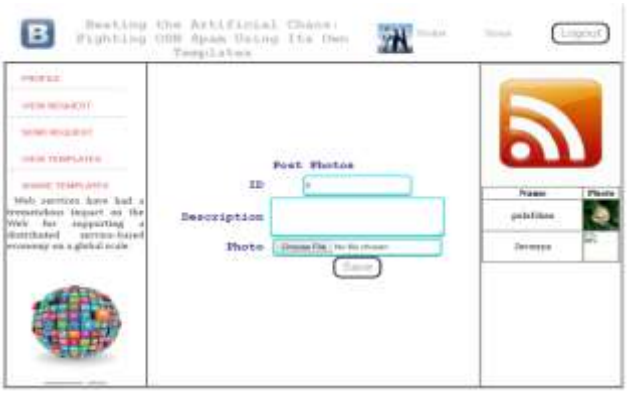

Fig4: Post Photos

Fig5: View and confirm request

Fig6: Admin Zone

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

Fig6: View Members in Admin Zone

Fig7: View Templates in Admin Zone

Fig8: Spam Box In Admin Zone

III. RESULTS AND DISCUSSION

We evaluate Tangram using the labeled dataset as ground truth. The two major metrics that we use to evaluate the system are accuracy and speed. For accuracy, we primarily evaluate the following two aspects:

i) true positive rate, the ratio of correctly classified messages to the total number of messages, (We only count spam caught by template matching or noise detection as true positives.) and ii) false positive rate, the ratio of legitimate but incorrectly classified messages to the total number of legitimate messages. (We count mis-

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

detected legitimate messages as false positives.) Besides, we may occasionally report false negatives, which count spam missed by the two modules but labeled in the ground truth. Defining the ratio of such messages to the total number of message in ground truth as false negative rate, we have that false negative rate + true positive rate = 1. For speed, we evaluate the template generation and matching latency. We feed the system with the collected messages obeying their timestamp order to reflect the performance in real-world scenario. We conduct all experiments on a server with an eight-core Xeon E5520 2.2GHz CPU and 16GB memory. Tangram needs an auxiliary spam filtering module to provide the initial set of spam messages to construct the underlying template. When a message reaches the auxiliary spam filter, it needs to be classified as either spam or legitimate with very little latency. We leverage an existing online OSN spam filtering tool to provide training spam samples toward conducting a realistic evaluation. We reuse the same parameters reported in the paper. The auxiliary spam filter is not an oracle. It may mistakenly report legitimate messages as spam, or miss to report spam messages. Note that in what follows we evaluate the detection accuracy of only the modules proposed in this paper, but not the accuracy of the auxiliary spam filter. We will factor the effect of auxiliary filter's accuracy to Tangram.

1. Detection Accuracy :

We test Tangram with spam window size t = 1000, which means when the number of spam messages that slip through the template matching module but are blocked by the auxiliary spam filter reaches 1000, the system will invoke the template generation module to infer the underlying templates of the messages. The value of parameter k is 4. The results show that the TP rate for the most prevalent template-based spam achieves 95.7%. The system can also detect some non–template-based spam messages, because the system treats all messages as if they were template-based, and makes best-effort detection. As expected, the TP rate of such messages is lower than that of template-based messages. The overall TP and FP rate are 76.2% and 0.12%, respectively.

2. True Positive Analysis:

Tangram has two detection modules. Both modules perform well on the specific spam category that they are designed for. The template generation/matching module can detect template-based spam with 95.7% TP rate (336,849 out of 558, 706 × 63%). The noise detection module can detect no-content spam with 73.8% TP rate (34,635 out of 558, 706 × 8.4%). Unfortunately, the true positive rate of the other two spam categories is lower. About 80% of the false negatives (spam misclassified as legitimate with a rate of 1 - TP rate) belong to the other two categories.

3. False Positive Analysis:

Since the labeling approach we use to build the ground truth may miss to label true spam tmessages ,We further compare the true positives against the detected tweets that are not labeled. We observe that spammers frequently attach messages marks (RT @username) and Mentions (@username) at the beginning of messages, as well as noise words after the embedded URL. Hence, we remove all the noise and acquire the stem of spam messages . Any messages that shares the same stem with spam messages is also regarded as spam. The comparison reveals that 15,271 (0.12%) messages reported by Tangram are neither labeled as spam, nor sharing the same stem with spam messages. They represent the false positives that our system incurs. We thus use it as a post-processing step in the evaluation, rather than adopting it in the system design. Among the false positive messages, 42.0% of them are caused by overly general spam templates. Another 21.7% of them are popular messages like birthday wishes for Nelson Mandela. These popular tweets are mistakenly reported as spam by the auxiliary filter, so templates are generated to match them.

4. Template Analysis:

To understand how spammers develop spam templates empirically in a realistic dataset. The actual URLs are replaced by the "{URL}" symbol. An ε means the dictionary macro may instantiated as an empty string. An "…" means that the dictionary macro has more options that are not shown due to readability and space consideration.

Size: We do not observe dominant templates in Twitter. The most popular template matches 11.1% of true positives. The top ten most popular templates match 50.5% of true positives.

5. Use of noise:

Although in theory spammers can insert noise at any position in the template, practical spam messages have noise at either the beginning or the end, or both. Noise at the beginning is usually Mentions (@username) and Retweet marks (RT @username). There can be multiple consecutive Mentions and Retweet marks at the

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

message beginning. Noise at the end contains primarily Hashtags (#XXX, XXX being a topic) and popular terms like celebrity names and TV shows.

6. Message diversity:

As the number of dictionary macro increases, the number of unique messages the template can generate increases quickly. multiple dictionary macros.

7. Detection Accuracy Comparison:

With Existing Work We limit the direct experimental comparison to only the approaches that examine the message content to detect spam. Syntactical Clustering + Machine Learning: We first compare with a recent spam detection work that adopts syntactical message clustering and supervised machine learning (denoted as the syntactical clustering approach hereafter) in detail. The two systems share similar design goals. In addition, the existing approach is used as the auxiliary spam filter in our experiment. Hence, it is crucial to quantify the detection accuracy gains over directly using the existing system. We run the system using the same dataset on which we test Tangram. The syntactical clustering approach achieves an overall detection accuracy of 63.3% TP rate and 0.27% FP rate. The true positive rate obtained by the syntactical clustering approach on our data is lower than the reported number .The reason is that our spam labeling approach labels more spam messages as ground truth, which the syntactical clustering approach does not detect. In contrast, Tangram achieves a substantial improvement on both the TP rate (to 76.2%) and the FP rate (to 0.12%). The difference between the spam detected by these two systems indicates that they can potentially complement each other. This simple integration suffers from increased FP rate of 0.33%, but can boost TP rate to 85.4%. Judo: To validate that our template generation technique is more tailored to OSN spam detection, we also compare our work with a recent email spam detection system called Judo. Judo detects email spam based on template generation. We have already presented the difference between the two systems analytically by elaborating the difference in the critical system assumptions, i.e., invariant substring in template and quality of training samples. We further demonstrate their difference in experimental results, column “Judo”. Different from our system, Judo requires training set that contains pure spam generated by the same underlying template. We implement the template generation mechanism of Judo as described in and test the detection accuracy using the same dataset. Even with small window size (10 spam messages), the generated templates can only achieve 35.9% TP rate. The TP rate further drops to 10.6% if the window size is increased to 20. On the other hand, the FP rate is high (5.0%). It shows that real-world OSN trace breaks the crucial assumptions of Judo. As a result, Judo achieves extremely high accuracy in email spam detection, but does not perform well for OSN spam detection.

8. Detection Accuracy Comparison With URL Blacklists:

It is well known that most spam fools people via deceitful (plain or shortened) URLs. Such URLs usually direct to websites that advertise scams, phish one’s credentials, or propagate other malicious contents. An intuitive countermeasure might detect spam using blacklisted URLs. If a messages contains a blacklisted URL, it is highly likely that the mesages is not benign. We verify such intuition on the following popular websites that provide URL blacklist query service—Bitly, Virustotal, and Wepawet. We collect a more recent dataset with 9 million messages from Twitter in October, 2014. Among spam messages therein,99.2% end with a shortened URL. We redirect all the URLs in the spam messages to see whether the landing page has been suspended. The results show that the ability of Bitly is limited—only 50.0% of spam tweets can be detected via suspended URLs on Bitsy. Then we perform similar URL checking against Virustotal and Wepawet. They respectively reveal 50.9% and 33.3% of the spam. Note that the detected spam messages via these websites are highly redundant; the overall detection accuracy is 58.1%. On the other hand, Tangram can detect them with the TP rate of as high as 77.9%. We run Tangram again over the January 2015 dataset. It yields a consistently satisfactory TP as in October 2014 dataset, which is 71.2%.

9. Effect of Auxiliary Filter Quality:

Since the auxiliary spam filter essentially provides training samples for template generation, it is crucial to understand how the accuracy of the auxiliary spam filter affects Tangram’s detection accuracy. As aforementioned, we focus on the accuracy of our proposed modules—template matching and noise detection. We first try to analytically capture how their accuracy varies with that of auxiliary filter. Let Ts and Tl represent the number of spam tweets and the number of legitimate tweets in the ground truth. Let Ss and Sl respectively denote the count of spam messages caught by template matching or noise detection and the count of legitimate messages mis-classified as spam. Then Ss+Sl denotes the count of all spam tweets detected by our modules of template matching and noise detection, for which we have FP = Sl/Tl and FN = 1 – Ss/Ts.Since Tl and Ts are

invariants from ground truth, we next analyze how Sl and Ss and therefore FP and FN are affected by auxiliary filter's FPaux and TPaux. (Note that we have FNaux = 1 − TPaux.). It further falls into two parts: one is legitimate messages in Tl matched by templates generated from FPauxTl messages the other is legitimate messages in Tl matched by templates generated from TPauxTs messages. As previous results show, the second part should be small. It is thus FPaux that affects more on our modules' FP. The larger FPaux is, the larger FP tends to be. Since it is hard to deduce the exact number of legitimate messages in Tl matched by templates generated from FPauxTl messages, we choose to empirically evaluate the effect of FPaux shortly.Although TPaux increases with the count of spam messages caught by auxiliary filter, more detected spam may not promise more templates. Template matching thus may not match and detect more spam than it does using generated templates. Again, we will empirically evaluate how TPaux affects our modules' accuracy. First, we mimic low true-positive–rate auxiliary filter by sampling 50%, 20% and 10% of spam uniformly at random to feed template matching. The resulting true positive rate is 64.9%, 64.9% and 63.0%, respectively. Given the randomness of tweet stream, such sampling decreases the number of spam tweets but not their generating templates. Our methods' true positive rate does not drop that much. We set auxiliary filter to have 0.5%, 1% and 2% false positive rate. The resulting false positive rate is 1.2%, 3.53% and 3.3%, respectively. Second, we choose Virus Total as auxiliary filter to further evaluate how our methods' accuracy (especially true positive rate) varies. Virus total's true positive rate is 50.9%. Using it as auxiliary filter, our methods yield a true positive rate of 57.7%. This is even lower than what we obtain using the above auxiliary filter with true positive rate of 10%. The reason is Virus total completely misses detecting spam messages generated by certain templates. Besides, our methods' false positive rate is 7.6% while Virustotal's is 4.3%. In summary, Tangram's true positive rate drops only marginally if the auxiliary filter has low true positive rate that makes spam messages with certain templates completely undetected. It is more sensitive to false positives from the auxiliary filter. Hence, in practice Tangram needs an auxiliary filter with a low false positive rate. This is a reasonable requirement, since we can tune the auxiliary filter to be conservative in reporting spam.

10. Template Generation/Matching Speed Template Matching:

The template matching latency incurred by Tangram is minimal and is not noticeable to users. Figure 2 plots the minimum, 25% quantile, 75% quantile and maximum of the template matching time as a function of the number of generated templates. We observe a large variance of template matching latency, because the time consumed for regular expression matching highly depends on the text being matched. Nevertheless, the largest latency in the entire dataset is less than 80ms. The overall trend is that the template matching latency, shown by the boxes representing the 25% quantile and the 75% quantile, grows slowly with the number of templates. Even with more than one thousand templates, the median template matching latency is only 8ms.

11. Template Generation:

It is crucial to throttle spam campaigns at their early stage. Hence, we measure how fast templates can be generated. The time to generate template depends on the number of buffered spam messages. In our experiment, the mean template generation time is only 2.3 seconds. Although slower than template matching**,** template generation is not the bottleneck of Tangram, since template generation is performed in parallel with template matching.

12. Sensitivity for New Campaigns:

We take the five largest campaigns, one of which matches the template instantiated by spam, and evaluate how fast Tangram reacts to newly emerged spam. We randomly select a small percentage of messages from each campaign, and use them as training samples to generate the template. We vary the percentage of training samples from 0.05% to 0.5%. The remaining messages serve as the testing set. We measure the true positive rate as the percentage of the testing set that the generated template can match. We observe that all campaigns achieve almost 100% coverage even with only 0.15% of messages as training samples. Three campaigns have lower coverage when only 0.05% of messages are used to generate the template, because the system has not observed all possible values of dictionary macros due to insufficient training samples. Nonetheless, the coverage quickly climbs up to almost 100% when the percentage of training samples increases. The result indicates that when new spam campaigns emerge, the system can react quickly and generate effective templates to throttle them.

13. Accuracy on Facebook Data:

To test Tangram's generality on other OSNs, we collect 4.7 million comments from public Facebook pages generated from January 2012 to April 2013, and run Tangram on the Facebook data. We use two well-known

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

blacklists, Mcafee siteadvisor [27] and myWoT [28], to label the ground truth: any comments embedded with blacklisted URLs are labeled as spam. The dataset contains 6,337 spam comments embedded with blacklisted URLs. Tangram achieves 77.8% true positive rate and 0.08% false positive rate. In particular, the spam template matching module and the syntactic clustering module contribute 72.1% true positive rate and 64.6% true positive rate, respectively. This illustrates that our system yields promising results on other OSNs as well.

IV. CONCLUSION

We have proposed and evaluated Tangram, a template-based system for accurate and fast OSN spam detection. Our measurement study reveals Other proposed techniques include focusing on embedded URL information like redirection chains, DNS and WHOIS information and so on classifying URLs' landing ,and using sender's reputation Building sender profile features takes time and it is difficult to adopt for real-time detection. Few existing works can both do real-time detection and filter spam without URLs.

Spam Measurement

Thomas et al. examine a large corpus of suspended Twitter accounts in ,which provides rich knowledge on Twitter spammers that inspires our work from multiple aspects. Cao et al. design and implement a malicious account detection system called Synchro Trap which was able to reveal a lot of malicious accounts .Yang et al. design some more robust features to detect more Twitter spammers through in-depth analysis of the evasion strategies utilized by up-to-date Twitter spammers [37]. In addition, Grier et al. and Gao et al. discovered the popularity of compromised spamming accounts in Twitter and Facebook, respectively. Due to the different data collection method, most spamming accounts in our dataset are created by spammers. Yang et al. analyze the social network formed by spamming accounts and reveal different categories of legitimate accounts that follow spamming accounts [38]. Levchendo et al. and Kanich et al. study the monetization of spam campaigns [39], [40].

Signature Generation

The problem of spam template generation bears similarity with polymorphic worm signature generation .The worm signature generation is based on the assumption that polymorphic worm content contains invariant substrings, which is reasonable because some invariant bytes are crucial for successfully exploiting the vulnerability. However, this assumption is not solid in the context of spam detection, where spammers can express the same message using different expressions in human language. Our Twitter spam analysis supports this argument. that the majority of Twitter spam is likely to instantiate underlying templates. Based on the empirical findings, Tangram mainly employs template generation/matching to mitigate OSN spam. Tangram distinguishes from existing template generation work in that it can construct template in the absence of invariant substrings. Tangram detects OSN spam in real-time without a separate training phase. We further study several situational-awareness inference of spammers' strategy. The findings promise more features for detecting spam and spamming accounts.

V. REFERENCES

[1] H. Gao et al., "Spam ain't as diverse as it seems: Throttling OSN spam with templates underneath," in Proc. ACSAC, 2014, pp. 76–85.
[2] H. Gao, J. Hu, C. Wilson, Z. Li, Y. Chen, and B. Y. Zhao, "Detecting and characterizing social spam campaigns," in Proc. IMC, 2010, pp. 35–47.
[3] H. Gao, Y. Chen, K. Lee, D. Palsetia, and A. Choudhary, "Towards online spam filtering in social networks," in Proc. NDSS, 2012, pp. 1–16.
[4] S. Lee and J. Kim, "Warning Bird: Detecting suspicious URLs in Twitter stream," in Proc. NDSS, 2012, pp. 1–13.
[5] K. Thomas, C. Grier, J. Ma, V. Paxson, and D. Song, "Design and evaluation of a real-time URL spam filtering service," in Proc. IEEE Symp. S&P, May 2011, pp. 447–462.
[6] K. Thomas, C. Grier, V. Paxson, and D. Song, "Suspended accounts in retrospect: An analysis of Twitter spam," in Proc. IMC, 2011,pp. 243–258.
[7] J. Mottl, "Twitter acknowledges 23 million active users are actually bots," Tech Times, Aug. 2014[Online]. Available:http://tinyurl.com/l755bvm
[8] C. Kreibich et al., "Spamcraft
[9] C. Kreibich et al., "On the spam campaign trail," in Proc. LEET, vol. 8. 2008, pp. 1–9.
[10] A. Pitsillidis et al., "Botnet judo: Fighting spam with itself," in Proc. NDSS, Mar. 2010, pp. 1–19.

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

[11] Q. Zhang, D. Y. Wang, and G. M. Voelker, "DSpin: Detecting automatically spun content on the Web," in Proc. NDSS, 2014, pp. 1–16.
[12] C. Yang, R. Harkreader, and G. Gu, "Die free or live hard? Empirical evaluation and new design for fighting evolving Twitter spammers," in Proc. RAID, 2011, pp. 318–337.
[13] G. Stringhini, C. Kruegel, and G. Vigna, "Detecting spammers on social networks," in Proc. ACSAC, 2010, pp. 1–9.
[14] F. Benevenuto, G. Magno, T. Rodrigues, and V. Almeida, "Detecting spammers on Twitter," in Proc. CEAS, vol. 6. 2010, p. 12.
[15] A. H. Wang, "Don't follow me: Spam detection in Twitter," in Proc. Int. Conf. Secur. Cryptogr. (SECRYPT), Jul. 2010, pp. 1–10.
[16] Y. Xie, F. Yu, K. Achan, R. Panigrahy, G. Hulten, and I. Osipkov, "Spamming botnets: Signatures and characteristics," in Proc. SIGCOMM, 2008, vol. 38. no. 4, pp. 171–182.
[17] M. Egele, G. Stringhini, C. Kruegel, and G. Vigna, "COMPA: Detecting compromised accounts on social networks," in Proc. NDSS, 2013, pp. 1–17.
[18] J. Song, S. Lee, and J. Kim, "Spam filtering in Twitter using sender-receiver relationship," in Proc. RAID, 2011, pp. 301–317.
[19] What the Trend, accessed on Jan. 31, 2015. [Online]. Available: http://www.whatthetrend.com/
[20] A. Ritter, S. Clark, Mausam, and O. Etzioni, "Named entity recognition in tweets: An experimental study," in Proc. EMNLP, 2011, pp. 1524–1534.
[21] T. Jiang and M. Li, "On the approximation of shortest common supersequences and longest common subsequences," in Proc. ICALP, 1994, pp. 191–202.
[22] C. Grier, K. Thomas, V. Paxson, and M. Zhang, "@spam: The underground on 140 characters or less," in Proc. CCS, 2010, pp. 27–37.
[23] J. D. Lafferty, A. McCallum, and F. C. N. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in Proc. ICML, 2001, pp. 282–289.
[24] Unleash the Power of the Link, accessed on May 1, 2015. [Online].Available: https://bitly.com/
[25] VirusTotal, accessed on May 1, 2015. [Online]. Available: https://www.virustotal.com/
[26] Wepawet, accessed on May 1, 2015. [Online]. Available: https://wepawet.iseclab.org/
[27] McAfee SiteAdvisor, accessed on May 1, 2015. [Online]. Available: http://www.siteadvisor.com/
[28] Safe Browsing Tool | WOT (Web of Trust), accessed on May 1, 2015. [Online]. Available: http://www.mywot.com/
[29] B. Stone-Gross, T. Holz, G. Stringhini, and G. Vigna, "The underground economy of spam: A botmaster's perspective of coordinating large-scale spam campaigns," in Proc. LEET, 2011, p. 4.
[30] X. Jin, C. X. Lin, J. Luo, and J. Han, "A data mining-based spam detection system for social media networks," Proc. VLDB Endowment, 2011, vol. 4. no. 12, pp. 1458–1461.
[31] L. Zhuang, J. Dunagan, D. R. Simon, H. J. Wang, and J. D. Tygar, "Characterizing botnets from email spam records," in Proc. LEET, 2008, Art. no. 2.
[32] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond blacklists: Learning to detect malicious Web sites from suspicious URLs," in Proc. KDD, 2009, pp. 1245–1254.
[33] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Identifying suspicious URLs: An application of large-scale online learning," in Proc. ICML, 2009, pp. 681–688.
[34] D. S. Anderson, C. Fleizach, S. Savage, and G. M. Voelker, "Spamscatter: Characterizing Internet scam hosting infrastructure," in Proc. USENIX Secur., 2007, Art. no. 10.
[35] S. Hao, N. A. Syed, N. Feamster, A. G. Gray, and S. Krasser, "Detecting spammers with SNARE: Spatio-temporal network-level automatic reputation engine," in Proc. USENIX Secur., vol. 9. 2009, pp. 101–118.
[36] Q. Cao, X. Yang, J. Yu, and C. Palow, "Uncovering large groups of active malicious accounts in online social networks," in Proc. CCS, 2014, pp. 477–488.
[37] C. Yang, R. Harkreader, and G. Gu, "Empirical evaluation and new design for fighting evolving Twitter spammers," IEEE Trans. Inf. Forensics Security, vol. 8, no. 8, pp. 1280–1293, Aug. 2013.
[38] C. Yang, R. Harkreader, J. Zhang, S. Shin, and G. Gu, "Analyzing spammers' social networks for fun and profit: A case study of cyber criminal ecosystem on Twitter," in Proc. 21st Int. Conf. WWW, 2012, pp. 71–80.
[39] K. Levchenko et al., "Click trajectories: End-to-end analysis of the spam value chain," in Proc. IEEE Symp. S&P, May 2011, pp. 431–446.
[40] C. Kanich et al., "Spamalytics: An empirical analysis of spam marketing conversion," in Proc. CCS, 2008, pp. 3–14.

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

[41] Z. Li, M. Sanghi, Y. Chen, M.-Y. Kao, and B. Chavez, "Hamsa: Fast signature generation for zero-day polymorphic worms with provable attack resilience," in Proc. IEEE Symp. S&P, May 2006, pp. 30–47.
[42] J. Newsome, B. Karp, and D. Song, "Polygraph: Automatically generating signatures for polymorphic worms," in Proc. IEEE Symp. S&P, May 2005, pp. 226–241.
[43] S. Hao, N. A. Syed, N. Feamster, A. G. Gray, and S. Krasser, "Detecting spammers with SNARE: Spatio-temporal network-level automatic reputation engine," in Proc. USENIX Secur., vol. 9. 2009, pp. 101–118.
[44] Q. Cao, X. Yang, J. Yu, and C. Palow, "Uncovering large groups of active malicious accounts in online social networks," in Proc. CCS, 2014, pp. 477–488.
[45] C. Yang, R. Harkreader, and G. Gu, "Empirical evaluation and new design for fighting evolving Twitter spammers," IEEE Trans. Inf. Forensics Security, vol. 8, no. 8, pp. 1280–1293, Aug. 2013.
[46] C. Yang, R. Harkreader, J. Zhang, S. Shin, and G. Gu, "Analyzing spammers' social networks for fun and profit: A case study of cyber criminal ecosystem on Twitter," in Proc. 21st Int. Conf. WWW, 2012, pp. 71–80.
[47] K. Levchenko et al., "Click trajectories: End-to-end analysis of the spam value chain," in Proc. IEEE Symp. S&P, May 2011, pp. 431–446.
[48] C. Kanich et al., "Spamalytics: An empirical analysis of spam marketing conversion," in Proc. CCS, 2008, pp. 3–14.
[49] Z. Li, M. Sanghi, Y. Chen, M.-Y. Kao, and B. Chavez, "Hamsa: Fast signature generation for zero-day polymorphic worms with provable attack resilience," in Proc. IEEE Symp. S&P, May 2006, pp. 30–47.
[50] J. Newsome, B. Karp, and D. Song, "Polygraph: Automatically generating signatures for polymorphic worms," in Proc. IEEE Symp. S&P, May 2005, pp. 226–241..

CITE AN ARTICLE

Tabassum, T., Kauser, L., & Abid, S. (n.d.). BEATING THE ARTIFICIAL CHAOS: FIGHTING OSN SPAM USING ITS OWN UNDERLYING TEMPLATES BY TANGRAM. *INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY*, *7*(4), 667-680.