HACETTEPE UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING
BBM204
PROGRAMMING ASSIGNMENT #4

Subject	**:** Phishing Detection from URLs via Ternary Search Trees and N-grams
Advisor	**:** R.A. Ahmet Selman Bozkır
Submission Date	**:** 21.5.2020
Due	**:** 4.6.2020
Prog. Language	**:** Java (Eclipse will be used to evaluate)

1) INTRODUCTION

In this assignment, you are expected to create a phishing detection system by utilizing web page URLs as a source of information. What is more is you will extract discriminative character n-grams having various lengths such as 3, 4, and 5. During the implementation, you will also benefit from *Ternary Search Trees* for storing significant n-grams along with their weights which will be later used to classify whether a suspicious URL is *phishing* or *legitimate*. One good advantage this assignment is that your detection system will be free of machine learning employment. It is obvious that, the use of Machine Learning contributes to the prediction quality. However, your mission, should you choose to accept it involves achieving this difficult task without utilizing the machine intelligence.

By definition, phishing is a kind of cyberattack in which the fake websites mimicking to their legitimate counterparts are exploited to trick innocent users in order to access their private and sensitive information such as username, password and bank account numbers. This kind of private information is then employed to access personal accounts for various aims that generally results illegal financial loses [1]. To date, various approaches combatting with phishing attacks have been developed. Despite of these attempts, the number of phishing attacks is continuously growing and the techniques used by phishers are evolving. According to the 2017 4th quarter phishing attacks report of Anti Phishing Working Group (APWG), more than 700.000 unique phishing attacks have been recorded during 2017 [2]. Moreover, as of October 2017, 348 target brands have been phished by scammers. Furthermore, 2017 has witnessed the exponential increase in SSL certified phishing web pages as 32% of the new phishing web pages have been equipped with SSL certificates during the 2017. This findings clearly show that, the arms race between anti phishers and scammers is continuing and the techniques of scammers are evolving to evade phishing detection mechanisms [3].

2) BACKGROUND

In this section, you will learn the concepts which you will benefit from during the design and implementation of your assignment. Therefore, it is highly suggested for you to read this sub-section before going on.

2.1) Ternary Search Tries

String searching algorithms are very important studying area of String algorithms that try to find place or existence of key within larger text. There are known algorithms such as Knuth--Morris-Pratt algorithm, Boyer-Moore algorithms etc. These algorithms are generally based on making a single call. However, indexing may be a better method if searching multiple times in the same text. In other words, different approaches may need to be developed to search on stored data. The ternary search tree (TST) is a good alternative for getting rid of the cost of linkage for each character in the *trie* and eliminating the empty links. It is also an important advantage for other languages such as Turkish, as it is also more applicable for non--English languages. TST can be called as “ternary search trie” or “prefix tree” in some resources such as our course book.

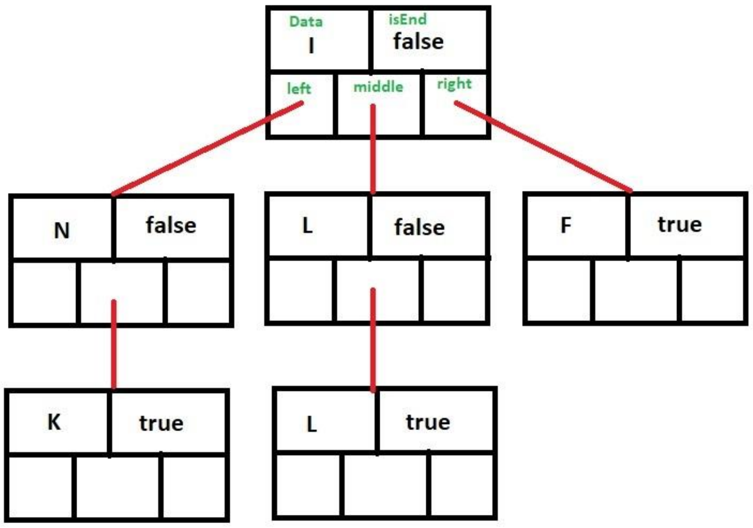


Fig 1. A sample TST involving 3 words namely “INK”, “ILL”, and “IF” adopted from [4]

In contrast to a standard *trie* data structure in which each node involves 26 pointers for its children, each node in a TST contains only 3 pointers:

- The left pointer points to the node whose value is less than the value in the current node.
- The equal pointer points to the node whose value is equal to the value in the current node.
- The right pointer points to the node whose value is greater than the value in the current node.

One benefit of employing TSTs over tries is that TSTs are a more space efficient (involve only 3 pointers per node rather than 26 which is in standard tries). Moreover, TSTs can be utilized as a hashtable in order to be used to store strings. In addition to these, each node has a field to store data (character in case of dictionary) and another field to mark end of a string.

2.2) Character N-grams

In the fields of computational linguistics and probability, an n-gram is a contiguous sequence of n items from a given sample of text or speech [5]. The items can be phonemes, syllables, letters, words or base pairs according to the application [5]. The n-grams typically are collected from a text or speech corpus. Using Latin numerical prefixes, an n-gram of size 1 is referred to as a "unigram"; size 2 is a "bigram" (or, less commonly, a "digram"); size 3 is a "trigram". English cardinal numbers are sometimes used, e.g., "four-gram", "five-gram", and so on.

For instance if we are dealing with a word such as “household” then its tri-grams become {“hou”, “ous”, “use”, “seh”, “eho”, “hol”, “old”}. Similarly, its four-grams are {“hous”, “ouse”, “useh”, “seho”, “ehol”, “hold”}. One advantage of using n-grams is that, they enable language independence when working on natural language processing (NLP) related tasks.

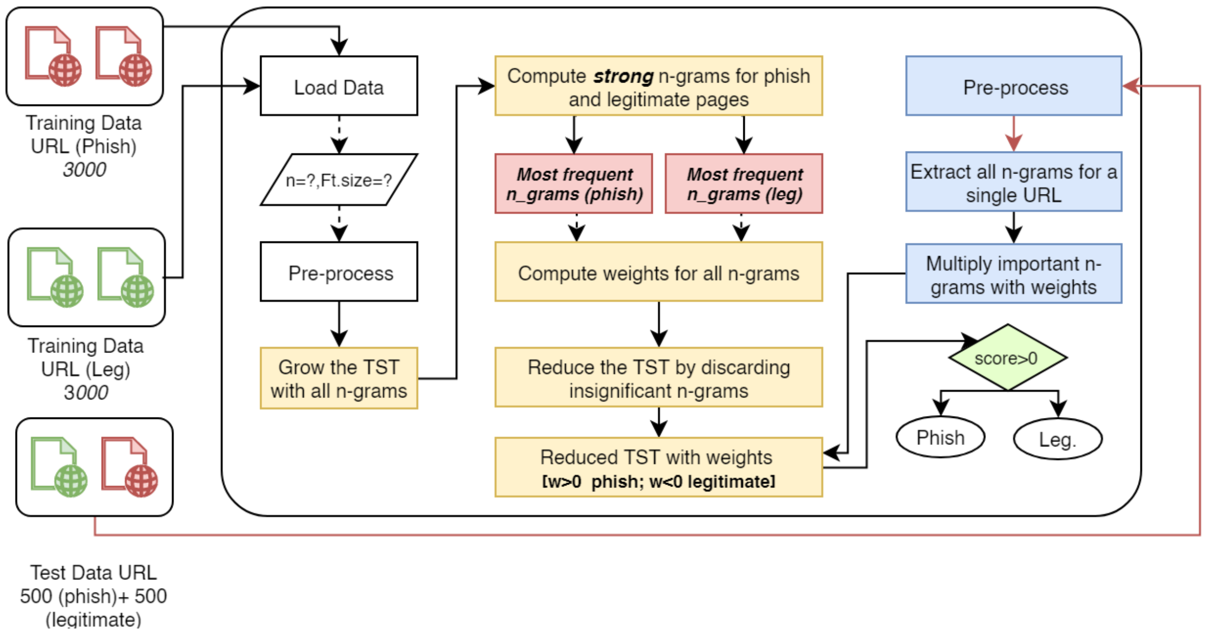

Training Data URL (Phish) 3000
Training Data URL (Leg) 3000
Test Data URL 500 (phish)+ 500 (legitimate)
Load Data
n=?,Ft.size=?
Pre-process
Grow the TST with all n-grams
Compute strong n-grams for phish and legitimate pages
Most frequent n_grams (phish)
Most frequent n_grams (leg)
Compute weights for all n-grams
Reduce the TST by discarding insignificant n-grams
Reduced TST with weights [w>0 phish; w<0 legitimate]
Pre-process
Extract all n-grams for a single URL
Multiply important n-grams with weights
score>0
Phish
Leg.

Fig 2. General workflow of the system. Some details regarding to file operations and reporting are not included in the schema

3) PROBLEM DEFINITION AND DETAILS

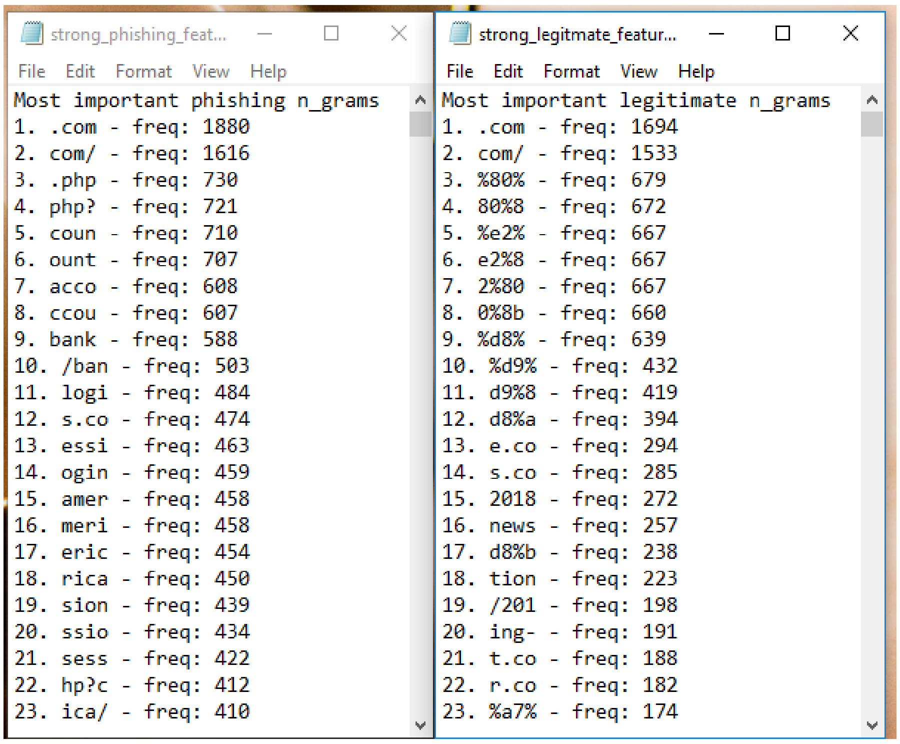
The main objective of this assignment is 5-folded. First of all, you will be provided with 3000 phishing and legitimate URLs categorized as “training” and “testing”. In other words, your application should first process 4 text files (i.e. “legitimate-train.txt“, ”legitimate-test.txt”, “phishing-train.txt” and “phishing-test.txt”). These files will include varying length web page URLs belonging to either phish or legitimate web pages.

Since most of the web page URLs start with either “https://” or “http://” , you must first discard these prefixes from the source URLs. Moreover, you also need to lowercase all the URL entries in both phishing and legitimate samples.

Following, you should extract each n-gram (suppose that we take n=4, four-grams) from each URL in training sets and build your ternary search tree by appending those n-grams into it. It is important that, apart from character data, your TST must also involve 3 other data-field for each node. These fields are “phish-occurrence”, “legitimate-occurrence” and “weight”. While you accumulate the TST, any n-gram may come from phishing or legitimate URL. If it is coming from phishing, you need to increment the field of “phish-occurrence” to denote that the n-gram has been seen in *phishing* environment. The same principle holds for the *legitimate* type.

Once all the training set (both phishing and legitimate training files are processed) your task is to write down all the most frequent n-grams found in training sets for phishing and legitimate side. Here, the importance is being adjusted by a user parameter called as “FEATURE_SIZE”. This is a start-up argument given by user in order to denote how many most frequent n-gram should be chosen. Note that, the argument of “FEATURE_SIZE” (i.e 1000) will be for phishing and legitimate based n-grams separately which yields computing 2000 (1000+1000) n-grams in total. The main reason for identifying these important n-grams is to reveal the significant and discriminative features for better classification.

In order to report these important/discriminative n-grams, you should create two separate text files named as “strong_phishing_features.txt” and “strong_legitimate_features.txt”. The sample file contents for most important phishing and legitimate n-grams have been depicted in Fig 3. As can be seen, the n-gram of ".com" takes the 1.st position for both phishing and legitimate pages. This shows that the same n-gram may be shared by both of these mediums. Meanwhile, it is mandatory to obey the textual format given in caption of Fig 3. Moreover, you must create all the output files in the parent folder of "src" directory. Similarly, your input files will also exist at there! Otherwise, your action will be punished with -10 points.



strong_phishing_feat...
File Edit Format View Help
strong_legitmate_featur...
File Edit Format View Help

```
Most important phishing n_grams
1. .com - freq: 1880
2. com/ - freq: 1616
3. .php - freq: 730
4. php? - freq: 721
5. coun - freq: 710
6. ount - freq: 707
7. acco - freq: 608
8. ccou - freq: 607
9. bank - freq: 588
10. /ban - freq: 503
11. logi - freq: 484
12. s.co - freq: 474
13. essi - freq: 463
14. ogin - freq: 459
15. amer - freq: 458
16. meri - freq: 458
17. eric - freq: 454
18. rica - freq: 450
19. sion - freq: 439
20. ssio - freq: 434
21. sess - freq: 422
22. hp?c - freq: 412
23. ica/ - freq: 410
```

```
Most important legitimate n_grams
1. .com - freq: 1694
2. com/ - freq: 1533
3. %80% - freq: 679
4. 80%8 - freq: 672
5. %e2% - freq: 667
6. e2%8 - freq: 667
7. 2%80 - freq: 667
8. 0%8b - freq: 660
9. %d8% - freq: 639
10. %d9% - freq: 432
11. d9%8 - freq: 419
12. d8%a - freq: 394
13. e.co - freq: 294
14. s.co - freq: 285
15. 2018 - freq: 272
16. news - freq: 257
17. d8%b - freq: 238
18. tion - freq: 223
19. /201 - freq: 198
20. ing- - freq: 191
21. t.co - freq: 188
22. r.co - freq: 182
23. %a7% - freq: 174
```

Fig 3. Example ordered contents for "strong_phishing_features.txt" and "strong_legitimate_features.txt". Note that the format of each line is "*#.item-number*. *n-gram* – freq: *frequency_of_n_gram*"

Next, you should compute the weight of all n-grams. In our solution, weight of an n-gram denotes the *characteristics* of it. In other words, the more it gets closer to +1, it denotes that the n-gram only appears in phishing URLs. In contrast, when it gets closer to -1, this indicates that, the n-gram appears more in legitimate URLs.

Here, we declare a formal definition for the computation of weights. Given that "*legitimate_occurence"* is denoted as LO whereas "*phishing_occurence"* is shown as PO:

```
if (PO>0 && LO==0)
     weight = 1  // a very unique n-gram for phishing medium
else if (PO ==0 && LO>0)
     weight = -1 // a very unique n-gram for legitimate medium
else if (PO >0 && LO>0)
{
    if (PO>LO)
        weight = min(PO,LO)/max(PO,LO) // (0,1)
    else if(PO<LO)
        weight = -min(PO,LO)/max(PO,LO) // (-1,0)
    else weight = 0 //the n-gram appears equally in both of the mediums
}
```

As can be inferred from the formula, each n-gram will function as a weight ranging from -1 to +1.

Once you compute the weights of all n-grams, write down them into a text file named **“all_feature_weights.txt”.** In listing below you can see the content and format of the file. For our case, it contains more than 119.000 n-grams along with their computed weights.

```
All N-Gram Weights
362a - weight: 1.0
xa=& - weight: 1.0
362c - weight: 1.0
362d - weight: 1.0
362e - weight: 1.0
3638 - weight: 1.0
3630 - weight: 1.0
3632 - weight: 1.0
3633 - weight: 1.0
1t36 - weight: 1.0

..
..
pdat - weight: 0.08045977
/ama - weight: 0.08
/bin - weight: 0.08
1756 - weight: 0.08
es/g - weight: 0.08
alid - weight: 0.078947365
verv - weight: 0.07874016
.125 - weight: 0.078431375
ache - weight: 0.07777778
..

..

36/s - weight: -1.0
_603 - weight: -1.0
_602 - weight: -1.0
3611 - weight: -1.0
3610 - weight: -1.0
3619 - weight: -1.0
363# - weight: -1.0
363/ - weight: -1.0
```

For the next process, after the identification of important n-grams, you need to remove unnecessary ones since this will lead you less computation for classification at next stages. Remember that, number of unique n-grams grow very fast. It is expected that, the number of candidate n-grams in this experiment will exceed 100.000 at extraction phase. Your task is to remove the insignificant ones, when the important ones get identified.

In a nutshell, we first grow the TST by accumulating all extracted n-grams, later we identify the most discriminative ones for both phishing and legitimate ones. Next, we compute the weights of each n-gram. Later, we remove the unnecessary n-grams from the TST. Keep in mind that, all these n-grams will function as a dictionary during the inference/classification stage. However, your functions and required fields must be in TST data structure. Using a hash-map for these operations must be limited! The main storage for all n-grams must be the TST itself.

Once you remove the unnecessary features/n-grams, then your TST must contain only significant features (i.e. for our example case – 1000 phish + 1000 legitimate **in ideal**). However, note that, the total number of significant features may be less than 2000 since some of them may be frequent enough to be elected but also shared between two categories**. In other words, an n-gram might be important for both of these categories along with being strong enough!**

The only remaining stage, after all, is to test the testing URLs found in “legitimate-test.txt” and “phishing-test.txt”. As stated before, the pre-process stage should be carried out for these files also. During the testing, you need to extract all n-grams for each URL and check whether your TST involves the newly extracted n-gram. In case it is found in your TST, accumulate the total score with the weight of related n-gram and compute overall sum for all significant n-grams found in the TST. In our experiment, the scheme for classification is defined as follows:

Suppose the URL*test* is composed of ***P*** n-grams. Since some of these will not be in TST during the reduction stage, we would detect only a subset of them namely ***S*** such that ***S*** <= ***P***. For all the significant n-grams found both in TST and URL*test*, take the sum of $S_{w,1}$, $S_{w,2}$, $S_{w,3}$... $S_{w,|S|}$. Finally, we either get a total_score higher than 0 (denotes it is phishing) or less than 0 (denotes legitimate). For some rare cases, we see that total_score gets equal to 0 which indicates that no meaningful prediction has been done for this case. For those cases we call them “Unpredictable”. Nonetheless, due to having ground-truth knowledge of incoming URLs, we can compute six different metrics:

- True Positive (TP): indicates the number of cases where actual URL is phishing and predicted is also true saying it is phishing.
- False Negative (FN): indicates the number of cases where actual URL is phishing but the prediction was wrongly made – saying it is legitimate.
- True Negative (TN): indicates the number of cases where actual URL is legitimate and the prediction is also true saying it is legitimate.
- False Positive (FP): indicates the number of cases where actual URL is legitimate but the prediction was wrongly made – saying it is phishing.
- Unpredictable Phishing (UP): indicates the number of cases where actual URL is phishing but computed total score is zero.
- Unpredictable Legitimate (UL): indicates the number of cases where actual URL is legitimate but computed total score is zero.

Based on these metrics you can compute the overall accuracy, given below:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN + UP + UL}$$

Your assignment will be based on console application. A sample output for 3000 training and 500 testing URLs (both for phishing and legitimate), n-gram: 4 and 1000 significant features (i.e. feature-size) for phishing and legitimate has been depicted in the listing below. As can be seen the prediction accuracy was obtained as 0.7315. Not bad ha?

```
n-gram based phishing detection via TST
feat_size: 1000
n_gram_size: 4

Legitimate training file has been loaded with [3000] instances
Legitimate test file has been loaded with [500] instances
Phishing training file has been loaded with [3000] instances
Phishing test file has been loaded with [500] instances
TST has been loaded with 3000 n-grams
TST has been loaded with 3000 n-grams
1000 strong phishing n-grams have been saved to the file"strong_phishing_features.txt"
1000 strong legitimate n-grams have been saved to the file
"strong_legitimate_features.txt"
114923 n-grams + weights have been saved to the file "all_feature_weights.txt"
113123 insignificant n-grams have been removed from the TST
TP:479 FN:21 TN:273 FP:227 Unpredictable Phishing:2 Unpredictable Legitimate:26
Accuracy: 0.7315175
```

Let us try a different *feature_size* for a better classification. For instance, let us try 5000 as the *feat_size*. The output has been given below. Now we have 0.8416 as the total accuracy. Much better!!!

```
n-gram based phishing detection via TST
feat_size: 5000
n_gram_size: 4

Legitimate training file has been loaded with [3000] instances
Legitimate test file has been loaded with [500] instances
Phishing training file has been loaded with [3000] instances
Phishing test file has been loaded with [500] instances
TST has been loaded with 3000 n-grams
TST has been loaded with 3000 n-grams
5000 strong phishing n-grams have been saved to the file"strong_phishing_features.txt"
5000 strong legitimate n-grams have been saved to the file
"strong_legitimate_features.txt"
114923 n-grams + weights have been saved to the file "all_feature_weights.txt"
105860 insignificant n-grams have been removed from the TST
TP:474 FN:26 TN:382 FP:118 Unpredictable Phishing:2 Unpredictable Legitimate:15
Accuracy: 0.841691
```

Notice that, how the running speed changes across different *feat_size* parameter. How about changing the n-gram size? Let us try changing the n value from 4 to 3 while keeping the feat_size as 5000. Wouw, we have increased the accuracy to 0.8642. You can tweak your parameters for achieving a better accuracy. **Now, congratulate yourself since you have an n-gram based phishing url detector without involving any kind of machine intelligence.**

```
n-gram based phishing detection via TST
feat_size: 5000
n_gram_size: 3

Legitimate training file has been loaded with [3000] instances
Legitimate test file has been loaded with [500] instances
Phishing training file has been loaded with [3000] instances
Phishing test file has been loaded with [500] instances
TST has been loaded with 3000 n-grams
TST has been loaded with 3000 n-grams
5000 strong phishing n-grams have been saved to the file"strong_phishing_features.txt"
5000 strong legitimate n-grams have been saved to the file
"strong_legitimate_features.txt"
33700 n-grams + weights have been saved to the file "all_feature_weights.txt"
25424 insignificant n-grams have been removed from the TST
TP:469 FN:31 TN:397 FP:103 Unpredictable Phishing:0 Unpredictable Legitimate:2
Accuracy: 0.86427146
```

4) GRADING POLICY

- Your work will be graded over a maximum of 100 points.
- There are five items you have to do in this experiment and the contribution of each item to your total score will be partial according to the grading policy stated below

Reading the file contents and building the TST with n-grams	15p
Computing the weights of n-grams and saving to a file	20p
Computing significant features and saving to separate files	25p
Classification of testing URLs	20p
Code design, algorithmic perspective, comments	20p

NOTES AND RESTRICTIONS:

- Use Eclipse platform for development
- Your experiment will be tested on a Windows PC equipped with Eclipse platform. During the assessment, various *n-gram* sizes and *feature_size* values will be tested also. Pay attention not to crash at first phase.
- The file names in this document are static and will not change. So, prepare your code by considering this information.
- Your experiment should be submitted before the due date via submit system. Late submissions will be penalized according to the previously announced rules.
- All assignments must be done individually unless stated otherwise. You are encouraged to discuss with your classmates about the given assignments, but these discussions should be carried out in an abstract way. That is, discussions related to a particular solution to a specific problem (either in actual code or in the pseudo code) will not be tolerated.
- In short, turning in someone else's work, in whole or in part, as your own will be considered as a violation of academic integrity. Please note that the former condition also holds for the material found on the web as everything on the web has been written by someone else.
- Use EXPLANATORY COMMENTS in your source codes.
- CAUTION: Do not start to thinking about assignment lately. Experiment requires less code but more algorithmic approach.
- You must send only two files, namely "TST.java" and "Main.java". Nothing more is required!
- You will submit your work from https://submit.cs.hacettepe.edu.tr with the file hierarchy as below:

```
-----(All the input files be located at parent folder of "src".
      |----src
          ----  Main.java
          ----  TST.java
```

REFERENCES:

[1] R. S. Rao and S. T. Ali, "A Computer Vision Technique to Detect Phishing Attack", 5th International Conference on Communication Systems and Network Technologies, pp. 596-601, 2015.

[2] APWG, Phishing Attack Trends Report 4th 2017, http://docs.apwg.org/reports/apwg_trends_report_q4_2017.pdf

[3] F.C. Dalgic, A.S. Bozkir and M. Aydos, "Phish-IRIS: A New Approach for Vision Based Brand Prediction of Phishing Web Pages via Compact Visual Descriptors", ISMSIT Kızılcahamam, Turkey, 2018.

[4] http://www.coderkt.co.in/2018/05/what-is-ternary-search-tree-ternary.html (Available online 28.8.2019)

[5] "What is a character n-gram?", https://www.wikizeroo.org/index.php?q=aHR0cHM6Ly9lbi53aWtpcGVkaWEub3JnL3dpa2kvTi1ncmFt (Available online 28.8.2019)