Spambots: Creative Deception

Authors:

Hayam Alamro & Costas S. Iliopoulos

Presenter: (Hayam Alamro)

- Department of Informatics, King's College London, UK
- Department of Information Systems, Princess Nourah bint Abdulrahman University, Riyadh, KSA.
- (hayam.alamro@kcl.ac.uk)

Short resume of the presenter :

- Hayam Alamro is a Ph.D. student in Computer Science (Algorithms & Data analysis Research Group) in the Department of Informatics at King's College London.
- Hayam's research focuses on the analysis and advanced design of string algorithms, approximate pattern matching, Cybersecurity, and data privacy.
- Hayam received her M.Sc. and her B.Sc. (with second class Honour) in Computer Science and Information Systems from King Saud University, Riyadh, Kingdom of Saudi Arabia.
- Before starting her Ph.D. in the UK, Hayam was working as a Lecturer in Computer Science and Information Systems College in Princess Nora University, Riyadh, Kingdom of Saudi Arabia. Hayam also has an experience working as a Computer Assistance in the Ministry of Planning, Interest of Public Statistics, Riyadh, Kingdom of Saudi Arabia.

Outline

- Introduction
- Our contributions
- Background & Definitions
- Problems Definitions
- Preprocessing Stage
- Deception with Errors
- Deception with Disguised Actions and Errors
- Deception with Don’t Cares Actions
- Conclusion

Introduction

https://images.app.goo.gl/a5Yreu3X7MSCHmvU7

- A *bot* is a software application that is designed to do certain tasks

Useful bots

- Used by some companies and business owners to improve customer services and communications (*chatbots*).

Harmful bots

- Used by spammers to target websites/ servers (*spambots*).

Introduction

https://images.app.goo.gl/a5Yreu3X7MSCHmvU7

- A *spambot* is a computer program designed to do repetitive actions on websites, servers or social media communities.

Activities

- Carrying out certain attacks on websites/ servers.
- Involving irrelevant links to increase a website ranking in search engine results.
- Using web crawlers for planting unsolicited material.
- Collecting email addresses from different sources (phishing emails).

Introduction

https://images.app.goo.gl/a5Yreu3X7MSCHmvU7

Existing spam detection techniques

- ***Content-based*** *:* Inject repetitive keywords in meta tags to promote a website in search engines.
- ***Link-based*** *:* Add links to a web page to increase its ranking score in search engines.
- ***Supervised machine learning:*** to identify the source of spambot, rather than detecting the spambot.

Nowdays: The spammers try to manipulate spambots' actions behaviour to appear as it were coming from a legitimate user to bypass the existing spam-filter tools

Introduction

https://images.app.goo.gl/a5Yreu3X7MSCHmvU7

Digital Creativity

- This work falls under the name of ***digital creativity*** where the *http* requests at the user log can be represented as temporally annotated sequences of actions.
- This representation helps explore repeated patterns of malicious actions with different variations of interleaving legitimate actions and malicious actions with variable time delays that the spammer might resort to deceive and bypass the existing spam detection tools.

Introduction

https://images.app.goo.gl/a5Yreu3X7MSCHmvU7

More relevant to our work

- ***String pattern matching-based techniques*** **(Hayati et al. and Ghanaei et al.),**

  But:

- They are inapplicable because they do not take into account temporal information of neither the sequence (i.e., the user log) nor the pattern (i.e., the spambot actions).

- P. Hayati, V. Potdar, A. Talevski, and W. Smyth, "Rule-based on-the-fly web spambot detection using action strings," in CEAS, 2010.
- V. Ghanaei, C. S. Iliopoulos, and S. P. Pissis, "Detection of web spambot in the presence of decoy actions," in IEEE International Conference on Big Data and Cloud Computing, 2014, pp. 277–279.

Our Contributions

- Provide a summary of the creativity of the spambot programmers, and the most important creative techniques that the spammer might use to deceive current spambot detection tools.
- Present our proposed solutions at this field to tackle those problems, including:

➢ Deception with Errors.

➢ Deception with Disguised Actions and Errors.

➢ Deception with Don’t Cares Actions.

Background & Definitions

- Let $T = a_0 \ldots a_{n-1}$ be a string of length $|T| = n$ over an alphabet Σ of size $|\Sigma| = \sigma$
- For $1 \leq i \leq j \leq n, T[i]$ denotes the ith symbol of T, and $T[i,j]$ the contiguous sequence of symbols (called *factor* or *substring*)
- A *substring* $T[i,j]$ is a ***suffix*** of T if $j = n$ and it is a ***prefix*** of T if $i = 1$
- A string p is a ***repeat*** of T iff p has at least two occurrences in T
- *A **degenerate or indeterminate string*** , is defined as a sequence $\tilde{X} = \tilde{x_0}\,\tilde{x_1} .. \widetilde{x_{n-1}}$, where $\tilde{x_i} \subseteq \Sigma$ for all $0 \leq i \leq n-1$
- A ***degenerate symbol*** $\tilde{x}$ over an alphabet Σ is a non-empty subset of Σ

Background & Definitions

- $|\tilde{x}|$ denotes the size of $\tilde{x}$, and we have $1 \leq \tilde{x} \leq |\Sigma|$.
- If $|\ \widetilde{x_i}\ | = 1$, that is $|\ \widetilde{x_i}\ |$ repeats a single symbol of Σ, we say that $\widetilde{x_i}$ is a ***solid symbol*** and i is a ***solid position***. Otherwise, $\widetilde{x_i}$ and i are said to be a ***non-solid symbol*** and ***non-solid position*** respectively.
- A ***conservative degenerate string*** is a degenerate string where its number of non-solid symbols is upper-bounded by a fixed position constant c.

Example

$$X = ab[ac]a[bcd]bac$$

Is a degenerate string of length 8 over the alphabet Σ = {a, b, c, d}, and conservative degenerate string with c = 2.

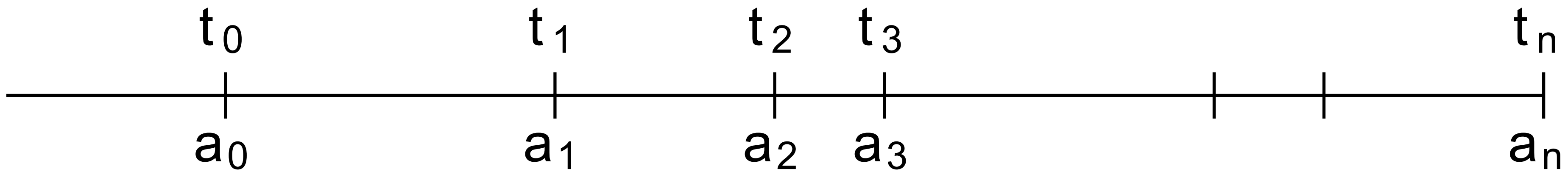
Background & Definitions

- A ***suffix array*** of T is the lexicographical sorted array of the suffixes of a string T i.e., the suffix array of T is an array SA[1...n] in which SA[i] is the ith suffix of T in ascending order.
- $\boldsymbol{LCP(T1, T2)}$ is the length of the longest common prefix between strings T_1 and T_2 , and it is usually used with SA such that $LCP[i]$ = lcp($T_{\mathrm{SA[i]}}, T_{\mathrm{SA[i-1]}}$) for all $i \in [1..n]$.
- The ***suffix tree T*** for a string *S* of length n over the alphabet Σ is a rooted directed compacted trie built on the set of suffixes of S.

Background & Definitions

Definition 1:

A Temporally Annotated Action Sequence: is a sequence $T = (a_0, t_0), (a_1, t_1), ..(a_n, t_n)$, where $a_i \in A$, with $\boldsymbol{A}$ set of actions, and $\boldsymbol{t_i}$ represents the time that action $\boldsymbol{a_i}$ took place. Note that $\boldsymbol{t_i} < \boldsymbol{t_{i+1}}$, ∀ i ∈ [0, n].



Background & Definitions

Definition 2:

An Action Sequence: is a sequence $(s_1 s_2 \ldots s_m)$ where $s_i \in A$, with $\boldsymbol{A}$ is the set of all possible actions.

Definition 3:

A Dictionary $\widehat{\boldsymbol{S}}$: is a collection of tuples $(S_{i,} W_i)$, where S_i is a temporally annotated sequence corresponding to a spambot and W_i is a time window (total estimated time for all set of actions performed by the spambot).

Background & Definitions

Definition 4:

Enhanced Suffix Array (ESA): is a data structure consisting of a suffix array and additional tables which can be constructed in linear time and considered as an alternative way to construct a *suffix tree* which can solve pattern matching problems in optimal time and space

Background & Definitions

Definition 5:

Generalized Enhanced Suffix Array (GESA): is an enhanced suffix array for a set of strings, each one ending with a special character and usually is built to find the *Longest Common Sequence (LCS)* of two strings or more.

GESA is indexed as a pair of identifiers (i_1, i_2).

String number

Lexicographical order of the suffix

Problems Definitions

1. **<u>Deception with Errors</u>**: Given a temporal annotated sequence $\boldsymbol{T} = (\boldsymbol{a_1}, \boldsymbol{t_1}) .. (\boldsymbol{a_n}, \boldsymbol{t_n})$, a dictionary $\widehat{\boldsymbol{S}}$ containing sequences $\boldsymbol{S_i}$, each associated with a time window $\boldsymbol{W_i}$, a minimum frequency threshold $\boldsymbol{f}$, and a maximum Hamming distance threshold $\boldsymbol{K}$, find all occurrences of each $\boldsymbol{S_i} \in \widehat{\boldsymbol{S}}$ **in** $\boldsymbol{T}$, such that each $\boldsymbol{S_i}$ occurs:

I. At least $\boldsymbol{f}$ times within its associated time window $\boldsymbol{W_i}$, and

II. With at most $\boldsymbol{K}$ mismatches according to Hamming distance.

Problems Definitions

2. **Deception with Disguised Actions and Errors**: Given a temporal annotated sequence $\boldsymbol{T} = (\boldsymbol{a_1}, \boldsymbol{t_1}) \ldots (\boldsymbol{a_n}, \boldsymbol{t_n})$, a dictionary $\overline{\boldsymbol{S}}$ containing sequences $\widehat{\mathbf{S_i}}$, each has a $\boldsymbol{c}$ non-solid symbol (represented by #), associated with a time window $\boldsymbol{W_i}$, a minimum frequency threshold $\boldsymbol{f}$, and a maximum Hamming distance threshold $\boldsymbol{K}$, find all occurrences of each $\widehat{\boldsymbol{S_i}} \in \overline{\boldsymbol{S}}$ **in** $\boldsymbol{T}$, such that each $\widehat{\boldsymbol{S_i}}$ occurs:

I. At least $\boldsymbol{f}$ times within its associated time window $\boldsymbol{W_i}$, and
II. With at most $\boldsymbol{K}$ mismatches according to Hamming distance.

Problems Definitions

3. **<u>Deception with Don't Cares Actions</u>**: Given a temporal annotated sequence $\boldsymbol{T} = (\boldsymbol{a_{1,}t_1}) \ldots (\boldsymbol{a_n, t_n}), \widehat{\boldsymbol{S}}$ containing sequences $\boldsymbol{S_i}$ over the alphabet $\Sigma \cup \{*\}$, each associated with a time window $\boldsymbol{W_i}$, a minimum frequency threshold $\boldsymbol{f}$, and a maximum Hamming distance threshold $\boldsymbol{K}$, find all occurrences of each $\boldsymbol{S_i} \in \widehat{\boldsymbol{S}}$ **in** $\boldsymbol{T}$, such that each $\boldsymbol{S_i}$ occurs:

I. At least $\boldsymbol{f}$ times within its associated time window $\boldsymbol{W_i}$, and

II. With at most $\boldsymbol{K}$ mismatches according to Hamming distance.

Preprocessing Stage

Our algorithms require as input sequences temporally annotated actions. These temporally annotated sequences are produced from ***user logs*** consisting of a collection of ***http requests***.

- each request in a user log is mapped to → a predefined index key + time point t

125.127.33.125 - smith [14/Oct/2019: 10:12:26 –0500] "GET /blog/page-address.htm HTTP/1.1" 200 1043 "-" "Mozilla/5.0 Chrome/80.0.3134.311"

Date – Time Stamp → **Time Point t**

Action Request → **Index Key**

Preprocessing Stage

Our Spambot Dictionary Representation

i	S_i	W_i (sec)
1	cbbx	20
2	byadc	25
…	…	…

Problem 1: Deception with Errors

- At this regard, the spammer is constantly trying to change the spambot actions behaviour to make it appear like human actions, either by replacing specific actions by others or changing the order of actions that causes errors in order to bypass the existing detection tools.
- Given a temporally annotated action sequence $\boldsymbol{T} = (\boldsymbol{a_0}, \boldsymbol{t_0})(\boldsymbol{a_1}, \boldsymbol{t_1}) \dots (\boldsymbol{a_{n-1}}, \boldsymbol{t_{n-1}})$, an action sequence $\boldsymbol{S_i} = \boldsymbol{s_1 s_2} \dots \boldsymbol{s_m}$, and an integer t, we will compute $\boldsymbol{j_1}, \boldsymbol{j_2}, \dots, \boldsymbol{j_m}$ such that $\boldsymbol{a_{j_i}} = \boldsymbol{s_i}, \boldsymbol{1} \leq \boldsymbol{i} \leq \boldsymbol{m}$ and $\sum_{i=1}^{m} \boldsymbol{t_{j_i}} < \boldsymbol{t}$ or $\boldsymbol{t_{j_m}} - \boldsymbol{t_{j_1}} < \boldsymbol{t}$ with **Hamming distance between** $\boldsymbol{T_a}$ and $\boldsymbol{S_i}$ no more than $\boldsymbol{k}$ **mismatches.**

Problem 1: Deception with Errors

Example

Suppose a spambot designed to promote the selling of products to the largest number of websites as a sequence of actions: *{User Registration, View Home Page, Start New Topic, Post a Comment on Products, View Topics, Reply to Posted Topic "with Buy Link"}* can be redesigned by replacing a few actions with others such that it does not affect the goal of the spambot as following: *{User Registration, View Home Page, Update a Topic, Post a Comment on Products, Preview Topic, Reply to Posted Comment "with Buy Link"}*.

Problem 1: Deception with Errors

Uncover Deception with Errors

- Our algorithm for solving (Problem 1) needs to perform pattern matching with k errors where each sequence S_i in $\hat{S}$ should occur in T at least f times within its associated time window W_i.
- For that, we employ an advanced data structure *Generalized Enhanced Suffix Array* (GESA) with the help of *Kangaroo* method.
- Our algorithm will refer to a collection of tables ($GESA$, $GESA^R$, LCS, T, $\widehat{S}_{S_i}$) ➔ to find deception spambots with errors within a time window t.

Problem 1: Deception with Errors

Uncover Deception with Errors

Step 1: Extract the actions of the temporally annotated sequence T into T_a such that it contains only the actions $\boldsymbol{a_1} \ldots \boldsymbol{a_n}$ from T.

Step 2: Build **Generalized Enhanced Suffix Array (GESA) for a collection of texts ($\boldsymbol{T_a}$** and $\boldsymbol{\widehat{S}}$**):**

$$GESA(T_a, \widehat{S}_{S_i}) = Ta!_0\, S_1!_1\, S_2!_2 \ldots S_r!_r$$

- S_1, .., S_r are spambots sequences belong to dictionary $\widehat{S}$
- $!_0$, .. , $!_r$ are special delimiters not in Σ and smaller than any alphabetical letter in $\boldsymbol{T_a}$.

Problem 1: Deception with Errors

Uncover Deception with Errors

- **Step 3:** For each S_i in the spambots dictionary $\widehat{S}_{S_i}$, the algorithm calculates the *Longest Common Sequence LCS* between $\boldsymbol{S_i}$ and $\boldsymbol{T_a}$ starting at position 0 in sequence $\boldsymbol{S_i}$ and at position j in sequence $\boldsymbol{T_a}$, such that the common substring starting at these positions is maximal as follows:

➢ Find the the suffix index $\boldsymbol{i}$ of $\boldsymbol{S_i}$ in $GESA$ using $\boldsymbol{GESA^R}$ table (retains all the lexicographical ranks of the suffixes of $GESA$).

➢ Find the closest suffix $\boldsymbol{j}$ (belongs to a sequence in $\boldsymbol{T_a}$) to the suffix $\boldsymbol{i}$ (of$\boldsymbol{S_i}$) in $GESA$**.**

Problem 1: Deception with Errors

Uncover Deception with Errors

➢ **Step 4:** Compute the ***Longest Common Sequence LCS*** between $GESA(i)$ and $GESA(j)$ as follows:

$$\boldsymbol{LCS\ (S_i, T_a) = max(LCP(}GESA(i_1, i_2), GESA(j_1, j_2)\boldsymbol{))} = \boldsymbol{l_0}$$

Where $\boldsymbol{l_0}$ is the maximum length of the *longest common subsequence* matching characters between $GESA(i_1, i_2)$ and $GESA(j_1, j_2)$ until the first mismatch (or one of the sequences terminates).

Problem 1: Deception with Errors

Uncover Deception with Errors

- Next, find the length of the *longest common subsequence* matching characters after previous mismatch position $\boldsymbol{l_0}$ using ***Kangaroo** method* until the second mismatch (or one of the sequences terminates) as follows:

$$\boldsymbol{max}(\boldsymbol{LCP}(GESA(i_1, i_2 + l_0 + 1), GESA(j_1, j_2 + l_0 + 1))) = \boldsymbol{l_1}$$

- The algorithm will continue to use the Kangaroo method to find the other number of mismatches, until the number of errors is greater than k or one of the sequences terminates .

Problem 1: Deception with Errors

Uncover Deception with Errors

- **Step 5:** Finally, at each occurrence of $\boldsymbol{S_i}$ in the sequence $\boldsymbol{T_a}$, our algorithm checks its time window W_i using the dictionary $\widehat{S}_{S_i}$ in $\boldsymbol{T}$, such that it sums up each time t_i associated with its action a_i starting at the position j_2 in $GESA(j_1, j_2)$ until the length of the spambot $|\boldsymbol{S_i}|$, and then compare it to its time window W_i. If the resultant time is less than or equal to W_i, the algorithm considers that the sequence $\boldsymbol{S_i}$ is spambots and terminates.

➢ The algorithm will continue to find other occurrences of the spambot sequence $\boldsymbol{S_i}$ using the adjacent suffixes to the suffix index of $\boldsymbol{S_i}$ in GESA.

Illustration by Example

Example

- Suppose : $T_a = abcabxabcdcabxab$
- $S_1 = cbbx$ $\qquad$ $S_2 = byadc$
- $K = 2, f = 2$
- Concatenation strings of $Ta\$_0S_1\$_1S_2\$_2$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
a	b	c	a	b	x	a	b	c	d	c	a	b	x	a	b	$\$_0$	c	b	b	x	$\$_1$	b	y	a	d	c	$\$_2$

Illustration by Example

i	$GESA[i]$	Suffix	$GESA^R[i]$
0	(0,16)	$\$_0 cbbx\$_1 byadc\$_2$	4
1	(1,21)	$\$_1 byadc\$_2$	11
2	(2,27)	$\$_2$	19
3	(0,14)	$ab\$_0 cbbx\$_1 byadc\$_2$	7
4	(0,0)	$abcabxabcdcabxab\$_0 cbbx\$_1 byadc\$_2$	15
5	(0,6)	$abcdcabxab\$_0 cbbx\$_1 byadc\$_2$	26
6	(0,11)	$abxab\$_0 cbbx\$_1 byadc\$_2$	5
7	(0,3)	$abxabcdcabxab\$_0 cbbx\$_1 byadc\$_2$	12
8	(2,24)	$adc\$_2$	21
9	(0,15)	$b\$_0 cbbx\$_1 byadc\$_2$	23
10	(1,18)	$bbx\$_1 byadc\$_2$	18
11	(0,1)	$bcabxabcdcabxab\$_0 cbbx\$_1 byadc\$_2$	6
12	(0,7)	$bcdcabxab\$_0 cbbx\$_1 byadc\$_2$	14
13	(1,19)	$bx\$_1 byadc\$_2$	25
14	**(0,12)**	$bxab\$_0\ cbbx\$_1 byadc\$_2$	3
15	**(0,4)**	$bxabcdcabxab\$_0\ cbbx\$_1 byadc\$_2$	9
16	(2,22)	$byadc\$_2$	0
17	(2,26)	$c\$_2$	20
18	**(0,10)**	$cabxab\$_0\ cbbx\$_1 byadc\$_2$	10
19	**(0,2)**	$cabxabcdcabxab\$_0\ cbbx\$_1 byadc\$_2$	13
20	**(1,17)**	$cbbx\$_1\ byadc\$_2$	24
21	(0,8)	$cdcabxab\$_0 cbbx\$_1 byadc\$_2$	1
22	(2,25)	$dc\$_2$	16
23	(0,9)	$dcabxab\$_0 cbbx\$_1 byadc\$_2$	27
24	(1,20)	$x\$_1$	8
25	(0,13)	$xab\$_0 cbbx\$_1 byadc\$_2$	22
26	(0,5)	$xabcdcabxab\$_0 cbbx\$_1 byadc\$_2$	17
27	(2,23)	$yadc\$_2$	2

- There are two occurrences for $\boldsymbol{S_1}$ in $\boldsymbol{T}$ at $i = 18\ and\ 19$

Problem 2: Deception with Disguised Actions and Errors

- Spammers might attempt to deceive detection tools by creating more sophisticated sequences of actions in a creative way as their attempt to disguise their actions by varying certain actions and making some errors .

Example

a spambot takes the actions ABCDEF, then ACCDEF, then ABDDEF

can be described as ➔ A[BC][CD]DEF

The action [BC] and [CD] are variations of the same sequence

- A, C, D, E, F ➔ **(solid symbols)**
- [BC] [CD] ➔ **(indeterminate** or **non-solid symbols)**
- A[BC][CD]DEF ➔ **(degenerate string)**

Problem 2: Deception with Disguised Actions and Errors

Uncover Deception with Disguised Actions & Errors

- Our algorithm for solving (Problem 2) is similar to the steps of the Problem 1, but considering the following steps:

  - For each ***non-solid*** $\boldsymbol{s_j}$ occurring in degenerate sequence $\tilde{S} = s_1 s_2 \ldots s_m$, we substitute each s_j with '#' symbol, where '#' is not in Σ. Let $\widehat{\boldsymbol{S}}$ be the resulting pattern after substitution.

$\tilde{S}$	A	[FG]	[CD]	D	B	E
$\widehat{S}$	A	$\#_1$	$\#_2$	D	B	E

Problem 2: Deception with Disguised Actions and Errors

- Our algorithm includes:

➢ Initialization for ***hashMatchTable*** to do ***bit masking*** operation.

➢ ***Kangaroo method*** to find the ***Longest Common Sequence LCS*** between a sequence of actions in $\boldsymbol{T}$ and an action sequence $\widehat{\boldsymbol{S}}_{\boldsymbol{i}}$ with at most $\boldsymbol{K}$ mismatches in linear time.

Problem 2: Deception with Disguised Actions and Errors

- Once our algorithm encounters '#' in the sequence $\widehat{S}_i$, it will get into the ***verification process:***
  - Query whether the corresponding action a_i (in T_a) belongs to the set of actions in '#', to do that:
  - The algorithm uses a ***bit masking*** operation (***And*** bit wise operation) between the two sets ('#' and a_i) such that (a_i is represented by a bit '1', and each action in '#' is represented by '1' and '0' otherwise using ***hashMatchTable***).

Problem 2: Deception with Disguised Actions and Errors

hashMatchTable

- The columns are indexed by the (ascii code) of each character ($\boldsymbol{a_i} \in \Sigma$) (for directly access)
- The rows are indexed by the number of the spambots sequence $\widehat{\boldsymbol{S}}_{\boldsymbol{i}}$ and the number of '$\#_l$'
- The algorithm applies the following formula:

$$1 \wedge hashMatchTable[\widehat{\boldsymbol{S}}_{\boldsymbol{r}}\#_{\boldsymbol{l}}][\text{ascii}[\boldsymbol{a_i}]]$$

Problem 2: Deception with Disguised Actions and Errors

hashMatchTable

$$\widetilde{S_1} = A[FG][CD]\,DBE \;\rightarrow\; \widehat{S}_1 = A\#_1\#_2DBE$$

Ascii(a_i)	65 A	66 B	67 C	68 D	… …	70 F	71 G	… …	88 X	89 Y	90 Z
$\widehat{S}_1\#_1$	0	0	0	0	…	1	1	…	0	0	0
$\widehat{S}_1\#_2$	0	0	1	1	…		0	…	0	0	0
…	…	…	…	…	…		..	…	…	…	…
$\widehat{S}_r\#_l$	…	…	…	…	…		..	…	…	…	…

Problem 3: Deception with Don’t Cares Actions

- This type of creative deception can be used when we do not care about the type of some actions, which appear between important actions in a sequence of actions.
- A don’t care action is denoted with ‘*’.
- **Example:** AB*D**H

Problem 3: Deception with Don’t Cares Actions

Uncover Deception with Don’t Cares Actions

- Our algorithm for solving (Problem 3) uses a fast and efficient algorithm which can locate all sequences of actions P with "don’t cares" of length m in text of actions S of length n in linear time, using ***suffix tree*** of S and ***Kangaroo*** method.

Problem 3: Deception with Don't Cares Actions

Uncover Deception with Don't Cares Actions

- Suppose we have this sequence $(P = AB * D *** H)$, and we want to locate all occurrences of pattern P in log of actions S.
- Our algorithm to solve it, using the following steps:

➢ Build the *suffix tree* of S

➢ Divide P into sub-patterns P_k: $P_1 P_2 P_3 = (AB *)(D\ * **)(H)$

➢ Using the *suffix tree* of S and the *Kangaroo* method which can be applied to selected suffixes of the suffix tree of S by the use of a predefined computational method to answer subsequent queries in O(k) time.

Conclusion

- We have presented our proposed algorithms that can detect a series of malicious actions which occur with variable time delays, repeated multiple times, and interleave legitimate and malicious actions in a creative way.
- Our proposed solutions tackled different types of creative deception that spammers might use to defeat existing spam detection techniques such as using errors, disguised, and "don’t cares" actions.
- Our algorithms took into account the temporal information because they considered time annotated sequences and required a match to occur within a time window.
- The algorithms solved the problems exactly in linear time and space, and they employed advanced data structures to deal with problems efficiently.

Thank You