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
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).
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

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.

**Nowadays**: The spammers try to manipulate spambots’ actions behaviour to appear as if it were coming from a legitimate user to bypass the existing spam-filter tools.
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.
More relevant to our work

- **String pattern matching-based techniques** ([Hayati et al.](https://images.app.goo.gl/a5Yreu3X7MSCHmvU7) and [Ghanaei et al.](https://images.app.goo.gl/a5Yreu3X7MSCHmvU7)),

  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).

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.
Let $T = a_0 \ldots a_{n-1}$ be a string of length $|T| = n$ over an alphabet $\Sigma$ of size $|\Sigma| = \sigma$

For $1 \leq i \leq j \leq n$, $T[i]$ denotes the $i$th 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 $\tilde{X} = \tilde{x}_0 \tilde{x}_1 \ldots \tilde{x}_{n-1}$, where $\tilde{x}_i \subseteq \Sigma$ for all $0 \leq i \leq n - 1$

A degenerate symbol $\tilde{x}$ over an alphabet $\Sigma$ is a non-empty subset of $\Sigma$
|\tilde{x}| \text{ denotes the size of } \tilde{x}, \text{ and we have } 1 \leq |\tilde{x}| \leq |\Sigma|.

If \(|\tilde{x}_i| = 1\), that is \(|\tilde{x}_i|\) repeats a single symbol of \(\Sigma\), we say that \(\tilde{x}_i\) is a \textit{solid symbol} and \(i\) is a \textit{solid position}. Otherwise, \(\tilde{x}_i\) and \(i\) are said to be a \textit{non-solid symbol} and \textit{non-solid position} respectively.

A \textit{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 \(\Sigma = \{a, b, c, d\}\), and conservative degenerate string with \(c = 2\).
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.

$LCP(T_1,T_2)$ is the length of the longest common prefix between strings $T_1$ and $T_2$, and it is usually used with SA such that $LCP[i] = lcp(T_{SA[i]}, T_{SA[i-1]})$ 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.
Definition 1:

A Temporally Annotated Action Sequence: is a sequence

\[ T = (a_0, t_0), (a_1, t_1), \ldots (a_n, t_n), \] where \( a_i \in A \), with \( A \) set of actions, and \( t_i \) represents the time that action \( a_i \) took place. Note that \( t_i < t_{i+1}, \forall i \in [0, n] \).
An Action Sequence: is a sequence \((s_1 s_2 \ldots s_n)\) where \(s_i \in A\), with \(A\) is the set of all possible actions.

A Dictionary \(\widehat{S}\): 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).
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.
**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 \((i_1, i_2)\).
1. **Deception with Errors**: Given a temporal annotated sequence \( T = (a_1, t_1) \ldots (a_n, t_n) \), a dictionary \( \hat{S} \) containing sequences \( S_i \), each associated with a time window \( W_i \), a minimum frequency threshold \( f \), and a maximum Hamming distance threshold \( K \), find all occurrences of each \( S_i \in \hat{S} \) in \( T \), such that each \( S_i \) occurs:

I. At least \( f \) times within its associated time window \( W_i \), and

II. With at most \( K \) mismatches according to Hamming distance.
2. **Deception with Disguised Actions and Errors**: Given a temporal annotated sequence $T = (a_1, t_1) \ldots (a_n, t_n)$, a dictionary $\mathcal{S}$ containing sequences $\mathcal{S}_i$, each has a non-solid symbol (represented by #), associated with a time window $W_i$, a minimum frequency threshold $f$, and a maximum Hamming distance threshold $K$, find all occurrences of each $\hat{S}_i \in \mathcal{S}$ in $T$, such that each $\hat{S}_i$ occurs:

I. At least $f$ times within its associated time window $W_i$, and

II. With at most $K$ mismatches according to Hamming distance.
3. **Deception with Don’t Cares Actions**: Given a temporal annotated sequence $T = (a_1, t_1) \ldots (a_n, t_n)$, $\hat{S}$ containing sequences $S_i$ over the alphabet $\Sigma \cup \{\ast\}$, each associated with a time window $W_i$, a minimum frequency threshold $f$, and a maximum Hamming distance threshold $K$, find all occurrences of each $S_i \in \hat{S}$ in $T$, such that each $S_i$ occurs:

I. At least $f$ times within its associated time window $W_i$, and

II. With at most $K$ mismatches according to Hamming distance.
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 $t$.

**Example Request:**

```
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”
```
Our Spambot Dictionary Representation

<table>
<thead>
<tr>
<th>$i$</th>
<th>$S_i$</th>
<th>$W_i$ (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>cbbx</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>byadc</td>
<td>25</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
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 $T = (a_0, t_0)(a_1, t_1) \ldots (a_{n-1}, t_{n-1})$, an action sequence $S_i = s_1s_2 \ldots s_m$, and an integer $t$, we will compute $j_1, j_2, \ldots, j_m$ such that $a_{j_i} = s_i, 1 \leq i \leq m$ and $\sum_{i=1}^{m} t_{j_i} < t$ or $t_{j_m} - t_{j_1} < t$ with Hamming distance between $T_a$ and $S_i$ no more than $k$ 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"\}. 
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 $(GESA, GESA^R, LCS, T, \hat{S}_{S_i})$ to find deception spambots with errors within a time window $t$. 

Uncover Deception with Errors
Problem 1: Deception with Errors

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

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

$$GESA(T_a, \hat{S}_{S_{i}}) = T_{a_{0}} S_{1_{1}} S_{2_{2}} \ldots S_{r_{r}}$$

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

Uncover Deception with Errors
Problem 1: Deception with Errors

Uncover Deception with Errors

- **Step 3:** For each $S_i$ in the spambots dictionary $\hat{S}_{S_i}$, the algorithm calculates the *Longest Common Sequence LCS* between $S_i$ and $T_a$ starting at position $0$ in sequence $S_i$ and at position $j$ in sequence $T_a$, such that the common substring starting at these positions is maximal as follows:
  - Find the suffix index $i$ of $S_i$ in $GESA$ using $GESAR$ table (retains all the lexicographical ranks of the suffixes of $GESA$).
  - Find the closest suffix $j$ (belongs to a sequence in $T_a$) to the suffix $i$ (of $S_i$) 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:

\[
LCS(S_i, T_a) = \max(LCP(GESA(i_1, i_2), GESA(j_1, j_2))) = l_0
\]

Where \( l_0 \) 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 $l_0$ using Kangaroo method until the second mismatch (or one of the sequences terminates) as follows:

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

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 $S_i$ in the sequence $T_a$, our algorithm checks its time window $W_i$ using the dictionary $\widehat{S}_S_i$ in $T$, 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 $|S_i|$, 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 $S_i$ is spambots and terminates.

- The algorithm will continue to find other occurrences of the spambot sequence $S_i$ using the adjacent suffixes to the suffix index of $S_i$ in GESA.
Suppose: $T_a = abcabxabcabcdcabxab$

$S_1 = cbbx \quad S_2 = byadc$

$K = 2, f = 2$

Concatenation strings of $T_a S_0 S_1 S_1 S_2 S_2$
There are two occurrences for $S_1$ in $T$ at $i = 18$ and 19

<table>
<thead>
<tr>
<th>$i$</th>
<th>GESA[i]</th>
<th>Suffix</th>
<th>GESA$^R[i]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(0,16)</td>
<td>$s_0cbbs_1 byadc_2$</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>(1,21)</td>
<td>$s_1byadc_2$</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>(2,27)</td>
<td>$s_2$</td>
<td>19</td>
</tr>
<tr>
<td>3</td>
<td>(0,14)</td>
<td>$ab_0cbbs_1 byadc_2$</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>(0,0)</td>
<td>$abcabxabcabxabcabss_0cbbs_1 byadc_2$</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>(0,6)</td>
<td>$abcdcabxabss_0cbbs_1 byadc_2$</td>
<td>26</td>
</tr>
<tr>
<td>6</td>
<td>(0,11)</td>
<td>$abxabss_0cbbs_1 byadc_2$</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>(0,3)</td>
<td>$abxabcabxabss_0cbbs_1 byadc_2$</td>
<td>12</td>
</tr>
<tr>
<td>8</td>
<td>(2,24)</td>
<td>$adcs_2$</td>
<td>21</td>
</tr>
<tr>
<td>9</td>
<td>(0,15)</td>
<td>$bs_0cbbs_1 byadc_2$</td>
<td>23</td>
</tr>
<tr>
<td>10</td>
<td>(1,18)</td>
<td>$bbss_1byadc_2$</td>
<td>18</td>
</tr>
<tr>
<td>11</td>
<td>(0,1)</td>
<td>$bcabxabcabxabcabss_0cbbs_1 byadc_2$</td>
<td>6</td>
</tr>
<tr>
<td>12</td>
<td>(0,7)</td>
<td>$bcabxabcabss_0cbbs_1 byadc_2$</td>
<td>14</td>
</tr>
<tr>
<td>13</td>
<td>(1,19)</td>
<td>$bs_1byadc_2$</td>
<td>25</td>
</tr>
<tr>
<td>14</td>
<td>(0,12)</td>
<td>$bs_0cbbs_1 byadc_2$</td>
<td>3</td>
</tr>
<tr>
<td>15</td>
<td>(0,4)</td>
<td>$bxabcabxabcabss_0 cbbs_1 byadc_2$</td>
<td>9</td>
</tr>
<tr>
<td>16</td>
<td>(2,22)</td>
<td>$byadc_2$</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>(2,26)</td>
<td>$c_2$</td>
<td>20</td>
</tr>
<tr>
<td>18</td>
<td>(0,10)</td>
<td>$cabxabs_0 cbbs_1 byadc_2$</td>
<td>10</td>
</tr>
<tr>
<td>19</td>
<td>(0,2)</td>
<td>$cabxabcabxabcabss_0 cbbs_1 byadc_2$</td>
<td>13</td>
</tr>
<tr>
<td>20</td>
<td>(1,17)</td>
<td>$cbbs_1 byadc_2$</td>
<td>24</td>
</tr>
<tr>
<td>21</td>
<td>(0,8)</td>
<td>$edcabs_0cbbs_1 byadc_2$</td>
<td>1</td>
</tr>
<tr>
<td>22</td>
<td>(2,25)</td>
<td>$dc_2$</td>
<td>16</td>
</tr>
<tr>
<td>23</td>
<td>(0,9)</td>
<td>$dcabxabs_0cbbs_1 byadc_2$</td>
<td>27</td>
</tr>
<tr>
<td>24</td>
<td>(1,20)</td>
<td>$xs_1$</td>
<td>8</td>
</tr>
<tr>
<td>25</td>
<td>(0,13)</td>
<td>$xs_0cbbs_1 byadc_2$</td>
<td>22</td>
</tr>
<tr>
<td>26</td>
<td>(0,5)</td>
<td>$xabcabxabcabss_0cbbs_1 byadc_2$</td>
<td>17</td>
</tr>
<tr>
<td>27</td>
<td>(2,23)</td>
<td>$yadcs_2$</td>
<td>2</td>
</tr>
</tbody>
</table>
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 \( \Rightarrow \) (solid symbols)
- [BC] [CD] \( \Rightarrow \) (indeterminate or non-solid symbols)
- A[BC][CD]DEF \( \Rightarrow \) (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 $s_j$ occurring in degenerate sequence $\tilde{S} = s_1 s_2 \ldots s_m$, we substitute each $s_j$ with ‘#’ symbol, where ‘#’ is not in $\Sigma$. Let $\hat{S}$ be the resulting pattern after substitution.

<table>
<thead>
<tr>
<th>$\tilde{S}$</th>
<th>A</th>
<th>[FG]</th>
<th>[CD]</th>
<th>D</th>
<th>B</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\hat{S}$</td>
<td>A</td>
<td>#1</td>
<td>#2</td>
<td>D</td>
<td>B</td>
<td>E</td>
</tr>
</tbody>
</table>
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 \( T \) and an action sequence \( \hat{S}_i \) with at most \( K \) mismatches in linear time.
Once our algorithm encounters ‘#’ in the sequence $\tilde{S}_i$, 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

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

$$1 \wedge hashMatchTable[\widehat{S},\#]\text{[ascii}[a_i]]$$
**Problem 2: Deception with Disguised Actions and Errors**

\[
\widehat{S}_1 = A[FG][CD] DBE \quad \Rightarrow \quad \widehat{S}_1 = A\#_1\#_2 DBE
\]

<table>
<thead>
<tr>
<th>Ascii((a_i))</th>
<th>65</th>
<th>66</th>
<th>67</th>
<th>68</th>
<th>...</th>
<th>70</th>
<th>71</th>
<th>...</th>
<th>88</th>
<th>89</th>
<th>90</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\widehat{S}_1#_1)</td>
<td>0 0 0 0 ... 1 1 ... 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\widehat{S}_1#_2)</td>
<td>0 0 1 1 ... 0 ... 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>... ... ... ... ... ... ... ... ...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\widehat{S}_r#_l)</td>
<td>... ... ... ... ... ... ... ... ... ...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

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 \( P = AB \ast D \ast\ast\ast H \), 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 \( P_k: P_1 P_2 P_3 = (AB \ast)(D \ast\ast\ast)(H) \)
  - 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.
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