Write-Up
Strange Broadcast

Enunciado

Dois e-mails foram interceptados com o assunto "Broadcast" com a mesma data de envio e remetente. Era uma mensagem em branco com um anexo. Mas havia uma coisa estranha com ele: geralmente um broadcast é composto de várias mensagens iguais encaminhadas para diferentes destinatários, mas nesse caso cada anexo tinha um conteúdo distinto. Suspeitando de que na verdade esses arquivos guardassem a mesma informação, chamaram você, um hacker experiente, para ajudar a obter a informação oculta. Os dois arquivos estão abaixo:

- https://goo.gl/8gjB5z
- https://goo.gl/Gcoz1F

Dica: essa flag é **beeem grande**!!!

Resolução

Perceba duas informações muito importantes do enunciado:

- Os dois arquivos tem a mesma informação (a flag)
- E a flag é "**beeem grande**"

Para quem já conhece de algoritmos de strings, juntando essas duas informações fica claro que estamos falando do algoritmo LCS (Longest Common Substring), ou, traduzindo, a maior substring comum. Para quem não conhece sobre o assunto, uma busca rápida no google por exemplo por "long common string" (sem as aspas), encontraria como primeiro link o wikipedia do algoritmo, que contém, inclusive o algoritmo que deve ser utilizado:

```
function LCSubstr(S[1..r], T[1..n])
    L := array(1..r, 1..n)
    z := 0
    ret := {}
    for i := 1..r
        for j := 1..n
            if S[i] == T[j]
                if i == 1 or j == 1
                    L[i,j] := 1
                else
                    L[i,j] := L[i-1,j-1] + 1
                if L[i,j] > z
                    z := L[i,j]
                    ret := {S[i-z+1..i]}
                else
                if L[i,j] == z
                    ret := ret + {S[i-z+1..i]}
            else
                L[i,j] := 0
    return ret
```

Convertendo pra python e executando obtemos strings que não parecem com flags:

```
set(['', "'WL:", '3h%,l', '}b36', 'cVE9l', '1PrIl',
'qF6', '958i', ',#,#l', '1%|L', 'q8u6', 'N8<L',
'Zope', 'L:)D', '$j*6', ',u(Z', '<Z\\Z', 'qLZL',
'Ig]8l', 'N]}Z', ',01$l', 'Wk87', 'FW\\Z', 'jSN6',
'%|L7', 'WqLZ', '1[#Z', 'n:3Zl', '@@2Z', 'osJ',
']]<Z', 'RWqL', 'Q)4Ml', '2yii', '*2\\i', 'f2yi',
```

```
'u6\\6', 'q{F6', '(x|xl', 'sJ', '>]j4l', 'JqF6',
'L747', '6\\67', 'U]N;l', 'N;18l', 'c9P1l', "W'WL",
'0)iEl', 'e,uZl', '4sk6l', 'VE9l', "'C8e", "'jWi",
'Qw_9l', 'peR7', '-&}\\l', 'os', '$uO6', "'W: f1"])
```

Procurando pela sigla do algoritmo no google novamente "LCS algorithm", o primeiro link fala de um algoritmo de mesma sigla, mas com o nome ligeiramente diferente: **Longest Common Subsequence**. O que o diferencia do outro é que os caracteres não são necessariamente adjacentes:

`https://www.geeksforgeeks.org/longest-common-subsequence-dp-4/`

Esse site já tem aimplementação direto em python:

```
def lcs(X, Y, m, n):
    if m == 0 or n == 0:
       return 0
    elif X[m-1] == Y[n-1]:
       return 1 + lcs(X, Y, m-1, n-1)
    else:
       return max(lcs(X, Y, m, n-1), lcs(X, Y, m-1, n))
```

Ao executar, recebemos a seguinte mensagem de erro:

```
RuntimeError: maximum recursion depth exceeded in cmp
```

Justamente pela string ser extremamente grande. Para esse caso temos que usar a segunda implementação, dada pelo mesmo site, um pouco mais complicada mas muito mais eficiente tanto em tempo de execução quanto em número de chamadas a funções. Essa implementação usa um paradgma chamado **programação dinâmica**:

```
def lcs(X , Y):
    # find the length of the strings
    m = len(X)
    n = len(Y)

    # declaring the array for storing the dp values
    L = [[None]*(n+1) for i in xrange(m+1)]

    """Following steps build L[m+1][n+1] in bottom up fashion
    Note: L[i][j] contains length of LCS of X[0..i-1]
    and Y[0..j-1]"""
    for i in range(m+1):
        for j in range(n+1):
            if i == 0 or j == 0 :
                L[i][j] = 0
            elif X[i-1] == Y[j-1]:
                L[i][j] = L[i-1][j-1]+1
            else:
                L[i][j] = max(L[i-1][j] , L[i][j-1])

    # L[m][n] contains the length of LCS of X[0..n-1] & Y[0..m-1]
    return L[m][n]
```

Mas essa implementação não é ainda a desejada, pois essa retorna apenas o tamanho da string procurada (203 no caso), não ela em si. Para essa correção basta alterar um pouco:

```
def lcs(X , Y):
    # find the length of the strings
    m = len(X)
    n = len(Y)

    # declaring the array for storing the dp values
    L = [[None]*(n+1) for i in xrange(m+1)]

    for i in xrange(m+1):
        for j in xrange(n+1):
```

```
            if i == 0 or j == 0 :
                L[i][j] = ""
            elif X[i-1] == Y[j-1]:
                L[i][j] = L[i-1][j-1]+Y[j-1]
            elif len(L[i-1][j])>len(L[i][j-1]):
                L[i][j] = L[i-1][j]
            else:
                L[i][j] = L[i][j-1]

    return L[m][n]
```

Usando porém esse algoritmo apesar de imprimirmos uma string não parece nada com uma flag (e não é). O problema é que pode existir mais de uma subsequência comum com o mesmo tamanho. Procuremos então no google como encontrar todas elas "all longest commom seqences" e encontramos o seguinte site:

`https://www.geeksforgeeks.org/print-longest-common-sub-sequences-lexicographical-order/`

Novamente com o algoritmo, mas agora em C, o que não é problema, pois se trata de uma função simples que usa a que já definimos (a que retorna o número de caracteres) para o cálculo:

```
// C++ program to find all LCS of two strings in
// sorted order.
#include<bits/stdc++.h>
#define MAX 100
using namespace std;

// length of lcs
int lcslen = 0;

// dp matrix to store result of sub calls for lcs
int dp[MAX][MAX];

// A memoization based function that returns LCS of
// str1[i..len1-1] and str2[j..len2-1]
int lcs(string str1, string str2, int len1, int len2,
                                        int i, int j)
{
    int &ret = dp[i][j];

    // base condition
    if (i==len1 || j==len2)
        return ret = 0;

    // if lcs has been computed
    if (ret != -1)
        return ret;

    ret = 0;

    // if characters are same return previous + 1 else
    // max of two sequences after removing i'th and j'th
    // char one by one
    if (str1[i] == str2[j])
        ret = 1 + lcs(str1, str2, len1, len2, i+1, j+1);
    else
        ret = max(lcs(str1, str2, len1, len2, i+1, j),
                  lcs(str1, str2, len1, len2, i, j+1));
    return ret;
}

// Function to print all routes common sub-sequences of
// length lcslen
```

```
void printAll(string str1, string str2, int len1, int len2,
              char data[], int indx1, int indx2, int currlcs)
{
    // if currlcs is equal to lcslen then print it
    if (currlcs == lcslen)
    {
        data[currlcs] = '\0';
        puts(data);
        return;
    }

    // if we are done with all the characters of both string
    if (indx1==len1 || indx2==len2)
        return;

    // here we have to print all sub-sequences lexicographically,
    // that's why we start from 'a'to'z' if this character is
    // present in both of them then append it in data[] and same
    // remaining part
    for (char ch='a'; ch<='z'; ch++)
    {
        // done is a flag to tell that we have printed all the
        // subsequences corresponding to current character
        bool done = false;

        for (int i=indx1; i<len1; i++)
        {
            // if character ch is present in str1 then check if
            // it is present in str2
            if (ch==str1[i])
            {
              for (int j=indx2; j<len2; j++)
              {
                // if ch is present in both of them and
                // remaining length is equal to remaining
                // lcs length then add ch in sub-sequenece
                if (ch==str2[j] &&
                  lcs(str1, str2, len1, len2, i, j) == lcslen-currlcs)
                {
                  data[currlcs] = ch;
                  printAll(str1, str2, len1, len2, data, i+1, j+1, currlcs+1);
                  done = true;
                  break;
                }
              }
            }

            // If we found LCS beginning with current character.
            if (done)
                break;
        }
    }
}
```

Que convertida pra python e otimizada vira:

```
pd_lcs = dict()

def lcs(X , Y):
    global pd_lcs
```

```
    # find the length of the strings
    m = len(X)
    n = len(Y)

    if (X,Y) not in pd_lcs:

    # declaring the array for storing the dp values
        L = [[None]*(n+1) for i in xrange(m+1)]

        for i in xrange(m+1):
            for j in xrange(n+1):
                if i == 0 or j == 0:
                    L[i][j] = 0
                elif X[i-1] == Y[j-1]:
                    L[i][j] = L[i-1][j-1]+1
                else:
                    L[i][j] = max(L[i-1][j],L[i][j-1])

        pd_lcs[(X,Y)] = L[m][n]
    return pd_lcs[(X,Y)]
```

Juntando tudo e executando:

```
import re

def findAll(s,c):
    i0 = 0
    while s[i0:].find(c)>=0:
        yield i0+s[i0:].find(c)
        i0 += s[i0:].find(c)+1

pd_lcs = dict()

def lcs(X , Y):
    global pd_lcs
    # find the length of the strings
    m = len(X)
    n = len(Y)

    if (X,Y) not in pd_lcs:

    # declaring the array for storing the dp values
        L = [[None]*(n+1) for i in xrange(m+1)]

        for i in xrange(m+1):
            for j in xrange(n+1):
                if i == 0 or j == 0:
                    L[i][j] = 0
                elif X[i-1] == Y[j-1]:
                    L[i][j] = L[i-1][j-1]+1
                else:
                    L[i][j] = max(L[i-1][j],L[i][j-1])

        pd_lcs[(X,Y)] = L[m][n]
    return pd_lcs[(X,Y)]

data = []
lcslen = 0
ja = set()
```

```
def printAll(str1, str2, indx1, indx2, currlcs):
    global data
    global ja
    if data[:currlcs] in ja: return
    len1,len2 = len(str1),len(str2)

    sat = min(currlcs,len('CTF-BR{'))
    if data[:sat]!='CTF-BR{'[:sat]: return

    if currlcs == lcslen:
        print(data)
        open('flag.txt','a').write(data+'\n')
        return

    if indx1>=len1 or indx2>=len2: return

    print data[:currlcs],len(ja),(indx1, indx2, currlcs)
    ja.add(data[:currlcs])

    let_ini = (sat<len('CTF-BR{'))

    done = False
    if let_ini:
        it = [indx1+n for n in findAll(str1[indx1:],'CTF-BR{'[sat])]
    else:
        it = xrange(indx1,len1)
    ja_at = set()
    done = False
    for i in it:
        try:
            for j2 in findAll(str2[indx2:],str1[i]):
                j = j2+indx2
                if lcs(str1[i:], str2[j:]) == lcslen-currlcs:
                    data = data[:currlcs]+str1[i]+data[currlcs+1:]
                    printAll(str1, str2, i+1, j+1, currlcs+1)
        except Exception as e:
            print str(e),' - ',indx2,len(str2),i,len(str1)


X1 = open('file1','r').read()
X2 = open('file2','r').read()
data = lcs(X1,X2)*'_'
lcslen = lcs(X1,X2)
printAll(X1,X2,0,0,0)
```

Ainda teremos um problema: apesar que quase chegarmos à flag são muitas possibilidades (devido ao tamanho dela, que já descobrimos que é 203) então demoraria muito tempo. Ao executar esse código fica claro que as palavras são separadas por _, de forma que podemos descobrir a flag de palavra em palavra. Vamos modificar o código para escrever todas as possibilidades até a primeira palavra:

```
import re

def findAll(s,c):
    i0 = 0
    while s[i0:].find(c)>=0:
        yield i0+s[i0:].find(c)
        i0 += s[i0:].find(c)+1

pd_lcs = dict()
```

```
def lcs(X , Y):
    global pd_lcs
    # find the length of the strings
    m = len(X)
    n = len(Y)

    if (X,Y) not in pd_lcs:

    # declaring the array for storing the dp values
        L = [[None]*(n+1) for i in xrange(m+1)]

        for i in xrange(m+1):
            for j in xrange(n+1):
                if i == 0 or j == 0:
                    L[i][j] = 0
                elif X[i-1] == Y[j-1]:
                    L[i][j] = L[i-1][j-1]+1
                else:
                    L[i][j] = max(L[i-1][j],L[i][j-1])

        pd_lcs[(X,Y)] = L[m][n]
    return pd_lcs[(X,Y)]

data = []
lcslen = 0
ja = set()
n_palavras = 1
data_found = 'CTF-BR{'

def printAll(str1, str2, indx1, indx2, currlcs):
    global data
    global ja
    if data[:currlcs] in ja: return
    len1,len2 = len(str1),len(str2)

    sat = min(currlcs,len(data_found))
    if data[:sat]!=data_found[:sat]: return

    if data[:currlcs].count('_')>=n_palavras:
        print(data[:currlcs])
        open('flag.txt','a').write(data+'\n')
        return

    if currlcs == lcslen:
        print(data)
        open('flag.txt','a').write(data+'\n')
        return

    if indx1>=len1 or indx2>=len2: return

    print data[:currlcs],len(ja),(indx1, indx2, currlcs)
    ja.add(data[:currlcs])

    let_ini = (sat<len(data_found))

    done = False
    if let_ini:
        it = [indx1+n for n in findAll(str1[indx1:],data_found[sat])]
    else:
```

```
        it = xrange(indx1,len1)
    ja_at = set()
    done = False
    for i in it:
        try:
            for j2 in findAll(str2[indx2:],str1[i]):
                j = j2+indx2
                if lcs(str1[i:], str2[j:]) == lcslen-currlcs:
                    data = data[:currlcs]+str1[i]+data[currlcs+1:]
                    printAll(str1, str2, i+1, j+1, currlcs+1)
        except Exception as e:
            print str(e),' - ',indx2,len(str2),i,len(str1)


X1 = open('file1','r').read()
X2 = open('file2','r').read()
data = lcs(X1,X2)*'_'
lcslen = lcs(X1,X2)
printAll(X1,X2,0,0,0)
```

Executando para $n\_palavras = 1$, obtemos apenas uma que faz sentido em inglês:
CTF-BR{L0Ng3St_

Vamos então marcar ela como achada na variável $data\_found$ e mudar $n\_palavras$ para 2, ou de forma simplificada para os próximos passos:

```
data_found = 'CTF-BR{L0Ng3St_'
n_palavras = data_found.count('_')+1

data_found = 'CTF-BR{L0Ng3St_c0mMoN_'
n_palavras = data_found.count('_')+1

data_found = 'CTF-BR{L0Ng3St_c0mMoN_Su8S39u3nCe_'
n_palavras = data_found.count('_')+1

data_found = 'CTF-BR{L0Ng3St_c0mMoN_Su8S39u3nCe_iS_'
n_palavras = data_found.count('_')+1

data_found = 'CTF-BR{L0Ng3St_c0mMoN_Su8S39u3nCe_iS_83773r_'
n_palavras = data_found.count('_')+1

data_found = 'CTF-BR{L0Ng3St_c0mMoN_Su8S39u3nCe_iS_83773r_W17h_'
n_palavras = data_found.count('_')+1

data_found = 'CTF-BR{L0Ng3St_c0mMoN_Su8S39u3nCe_iS_83773r_W17h_DyN4m1C_'
n_palavras = data_found.count('_')+1

data_found = 'CTF-BR{L0Ng3St_c0mMoN_Su8S39u3nCe_iS_83773r_W17h_DyN4m1C_\
                Pr06r4Mm1N6_'
n_palavras = data_found.count('_')+1

data_found = 'CTF-BR{L0Ng3St_c0mMoN_Su8S39u3nCe_iS_83773r_W17h_DyN4m1C_\
                Pr06r4Mm1N6_80770m-uP_'
n_palavras = data_found.count('_')+1

data_found = 'CTF-BR{L0Ng3St_c0mMoN_Su8S39u3nCe_iS_83773r_W17h_DyN4m1C_\
                Pr06r4Mm1N6_80770m-uP_th3N_'
n_palavras = data_found.count('_')+1

data_found = 'CTF-BR{L0Ng3St_c0mMoN_Su8S39u3nCe_iS_83773r_W17h_DyN4m1C_\
                Pr06r4Mm1N6_80770m-uP_th3N_wi7h_'
n_palavras = data_found.count('_')+1
```

```
data_found = 'CTF-BR{L0Ng3St_c0mMoN_Su8S39u3nCe_iS_83773r_W17h_DyN4m1C_\
_____Pr06r4Mm1N6_80770m-uP_th3N_wi7h_R3curs10N_'
n_palavras = data_found.count('_')+1

\begin{lstlisting}[language=python]
data_found = 'CTF-BR{L0Ng3St_c0mMoN_Su8S39u3nCe_iS_83773r_W17h_DyN4m1C_\
_____Pr06r4Mm1N6_80770m-uP_th3N_wi7h_R3curs10N_70P-D0wn_'
n_palavras = data_found.count('_')+1

data_found = 'CTF-BR{L0Ng3St_c0mMoN_Su8S39u3nCe_iS_83773r_W17h_DyN4m1C_\
_____Pr06r4Mm1N6_80770m-uP_th3N_wi7h_R3curs10N_70P-D0wn_83cauS3_'
n_palavras = data_found.count('_')+1

\begin{lstlisting}[language=python]
data_found = 'CTF-BR{L0Ng3St_c0mMoN_Su8S39u3nCe_iS_83773r_W17h_DyN4m1C_\
_____Pr06r4Mm1N6_80770m-uP_th3N_wi7h_R3curs10N_70P-D0wn_83cauS3_\
_____oF_'
n_palavras = data_found.count('_')+1

data_found = 'CTF-BR{L0Ng3St_c0mMoN_Su8S39u3nCe_iS_83773r_W17h_DyN4m1C_\
_____Pr06r4Mm1N6_80770m-uP_th3N_wi7h_R3curs10N_70P-D0wn_83cauS3_\
_____oF_sup3R_'
n_palavras = data_found.count('_')+1

data_found = 'CTF-BR{L0Ng3St_c0mMoN_Su8S39u3nCe_iS_83773r_W17h_DyN4m1C_\
_____Pr06r4Mm1N6_80770m-uP_th3N_wi7h_R3curs10N_70P-D0wn_83cauS3_\
_____oF_sup3R_L0N6_'
n_palavras = data_found.count('_')+1

data_found = 'CTF-BR{L0Ng3St_c0mMoN_Su8S39u3nCe_iS_83773r_W17h_DyN4m1C_\
_____Pr06r4Mm1N6_80770m-uP_th3N_wi7h_R3curs10N_70P-D0wn_83cauS3_\
_____oF_sup3R_L0N6_f113S_'
n_palavras = data_found.count('_')+1

data_found = 'CTF-BR{L0Ng3St_c0mMoN_Su8S39u3nCe_iS_83773r_W17h_DyN4m1C_\
_____Pr06r4Mm1N6_80770m-uP_th3N_wi7h_R3curs10N_70P-D0wn_83cauS3_\
_____oF_sup3R_L0N6_f113S_As_'
n_palavras = data_found.count('_')+1

data_found = 'CTF-BR{L0Ng3St_c0mMoN_Su8S39u3nCe_iS_83773r_W17h_DyN4m1C_\
_____Pr06r4Mm1N6_80770m-uP_th3N_wi7h_R3curs10N_70P-D0wn_83cauS3_\
_____oF_sup3R_L0N6_f113S_As_th3Se_'
n_palavras = data_found.count('_')+1

data_found = 'CTF-BR{L0Ng3St_c0mMoN_Su8S39u3nCe_iS_83773r_W17h_DyN4m1C_\
_____Pr06r4Mm1N6_80770m-uP_th3N_wi7h_R3curs10N_70P-D0wn_83cauS3_\
_____oF_sup3R_L0N6_f113S_As_th3Se_4r3_'
n_palavras = data_found.count('_')+1

data_found = 'CTF-BR{L0Ng3St_c0mMoN_Su8S39u3nCe_iS_83773r_W17h_DyN4m1C_\
_____Pr06r4Mm1N6_80770m-uP_th3N_wi7h_R3curs10N_70P-D0wn_83cauS3_\
_____oF_sup3R_L0N6_f113S_As_th3Se_4r3_r3l473d_'
n_palavras = data_found.count('_')+1

data_found = 'CTF-BR{L0Ng3St_c0mMoN_Su8S39u3nCe_iS_83773r_W17h_DyN4m1C_\
_____Pr06r4Mm1N6_80770m-uP_th3N_wi7h_R3curs10N_70P-D0wn_83cauS3_\
_____oF_sup3R_L0N6_f113S_As_th3Se_4r3_r3l473d_70_'
n_palavras = data_found.count('_')+1
```

```
data_found = 'CTF-BR{L0Ng3St_c0mMoN_Su8S39u3nCe_iS_83773r_W17h_DyN4m1C_\
_____Pr06r4Mm1N6_80770m-uP_th3N_wi7h_R3curs10N_70P-D0wn_83cauS3_\
_____oF_sup3R_L0N6_f113S_As_th3Se_4r3_r3l473d_70_7h3_'
n_palavras = data_found.count('_')+1

data_found = 'CTF-BR{L0Ng3St_c0mMoN_Su8S39u3nCe_iS_83773r_W17h_DyN4m1C_\
_____Pr06r4Mm1N6_80770m-uP_th3N_wi7h_R3curs10N_70P-D0wn_83cauS3_\
_____oF_sup3R_L0N6_f113S_As_th3Se_4r3_r3l473d_70_7h3_r3cUr7i0N_'
n_palavras = data_found.count('_')+1

data_found = 'CTF-BR{L0Ng3St_c0mMoN_Su8S39u3nCe_iS_83773r_W17h_DyN4m1C_\
_____Pr06r4Mm1N6_80770m-uP_th3N_wi7h_R3curs10N_70P-D0wn_83cauS3_\
_____oF_sup3R_L0N6_f113S_As_th3Se_4r3_r3l473d_70_7h3_r3cUr7i0N_\
_____s7ACk_'
n_palavras = data_found.count('_')+1

data_found = 'CTF-BR{L0Ng3St_c0mMoN_Su8S39u3nCe_iS_83773r_W17h_DyN4m1C_\
_____Pr06r4Mm1N6_80770m-uP_th3N_wi7h_R3curs10N_70P-D0wn_83cauS3_\
_____oF_sup3R_L0N6_f113S_As_th3Se_4r3_r3l473d_70_7h3_r3cUr7i0N_\
_____s7ACk_M4x1Mum_'
n_palavras = data_found.count('_')+1
```

Temos, finalmente, a flag completa:

CTF-BR{L0Ng3St_c0mMoN_Su8S39u3nCe_iS_83773r_W17h_DyN4m1C_Pr06r4Mm1N6_80770m-uP_th3N_wi7h_R3curs10N_70P-D0wn_83cauS3_oF_sup3R_L0N6_f113S_As_th3Se_4r3_r3l473d_70_7h3_r3cUr7i0N_s7ACk_M4x1Mum_s1z3}iS7h3FLAG.