Australian Government

Cyber Security Challenge Australia 2014
www.cyberchallenge.com.au

CySCA2014 Mad Coding Skillz Writeup

Background: Fortcerts have tasked you with completing a number of programming tasks to assist them in the certification process for a number of customer software products.

Mad Coding Skillz 1 - Jeremy's Iron

Question: FortCerts needs you to write a program to test the functionality of a customers anagram program. Write a program that will unscramble the given word from a list of words and return it to the server. To be sure that the testing is reliable you will need to do this multiple times before the flag is revealed. The customer program is running at 172.16.1.20:5050

Designed Solution: Players need to receive a list of words and then receive a jumbled word. They then have to determine which word from the list of words it is and send it back to the server within the time limit. The intended way of doing this is to write a script to sort all of the list words and the jumbled word alphabetically and then use a string comparison to check for equality. Repeat this process until the flag is returned.

Write Up:
We used Python to write a script that receives the list of words and uses regex to extract the unjumbled words. It then receives the jumbled word, sorts all of the characters of each word alphabetically and compares them to find a match. Once a match has been found it sends the original unjumbled word back to the server. It repeats this process until the flag is returned.

```
#!/usr/bin/python
import socket
import sys
import re
```

```
# connect to server
s = socket.socket( socket.AF_INET, socket.SOCK_STREAM )
s.connect( ("172.16.1.20", 5050 ) )

def attempt():
       while True:
       words = s.recv(1000)
       if len(words) == 0:
              sys.exit(1)
       print( words[:-1] )
       matches = re.search( "Wordlist: (\[[^\]]*\])", words )
       if matches and matches.group( 1 ):
              l = eval( matches.group( 1 ) )
              sl = []
              for i,w in enumerate(l):
              sl.append( "".join( sorted( w ) ) )
              while True:
              jumbled = s.recv(1000)
              print( jumbled[:-1] )
              matches = re.search( "Jumbled word: ([a-z]+)", jumbled )
              if matches and matches.group( 1 ):
                     wordIndex = sl.index( "".join( sorted( matches.group( 1 ) ) ) )
)
                     s.send( l[ wordIndex ] + "\n" )
                     print( "sent: " +  l[ wordIndex ] )
                     return

if __name__ == "__main__":
       while True:
       attempt()
```

We run this script and receive the flag for this challenge. **IndianAttemptGermany771**

```
#> python soln.py
        Welcome to the jumbled word server.
        ==============================
[+] Unjumble 50 words sequentially within 60 seconds.
Wordlist: ['circumlocutions', 'unexceptionable', 'demonstratively',
'inconsiderately', 'instrumentality', 'aerodynamically', 'nationalization',
'correspondingly', 'parenthetically', 'improbabilities', 'physiotherapist',
'inconspicuously', 'superstitiously', 'extraordinarily', 'democratization',
'prohibitionists', 'disappointments', 'impressionistic', 'intellectualize',
'humanitarianism', 'conceptualizing', 'inconsistencies', 'unacceptability',
'interconnecting', 'paleontologists']

Jumbled word: cyorndyellaamia
sent: aerodynamically
**** SNIP ****
Wordlist: ['misappropriated', 'accomplishments', 'interchangeable',
'underestimating', 'trustworthiness', 'superintendents', 'conglomerations',
```

```
'segregationists', 'reconsideration', 'reconciliations', 'parenthetically',
'sentimentalized', 'disorganization', 'straightjackets', 'differentiation',
'hallucinogenics', 'indemnification', 'computationally', 'counterbalanced',
'psychotherapist', 'notwithstanding', 'bibliographical', 'bacteriological',
'demographically', 'unpronounceable']

Jumbled word: umtedetrannsiig
sent: underestimating
Enter unjumbled word:
```

IndianAttemptGermany771

Mad Coding Skillz 2 - Autobalance

Question: FortCerts is certifying a server program that generates challenges for authentication purposes. Write a program that will solve the challenges provided by the server. To ensure results are consistent and repeatable you will need to solve a number of challenges before you are authenticated and gain the flag. The server program is running at 172.16.1.20:9876

Designed Solution: This problem provides you with a set of positive and negative integers. You are required to find a subset of a specific length that sum to zero. This is variation of the classic "subset sum problem". Finding a solution to this problem is trivial - simply try every possible subset combination. The difficulty arises trying to calculate a number of these problems in a limited amount of time. Speedy solutions require a dynamic programming approach to remove repeated calculations.

Write Up:
Consider a simple set: { −7, −3, −2, 5, 8 }

A naive solution that attempts every combination will compute both of the following:

```
{ -7, -3, -2, 5 } and { -7, -3, -2, 8 }
```

Notice that there is repeated computation: -7, -3, and -2 are being summed in each combination. Whenever repeated computation is encountered there is an opportunity to store this result and re-use it, producing a more optimal solution.

Dynamic programming is a method of solving problems like this by breaking the problem down into a number of subproblems and solving each subproblem only once. There are a number of ways to accomplish this, the most popular memoization techniques among the testers were python dictionaries and matrices.

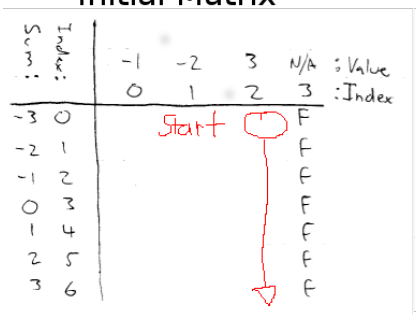
Below is a small hand worked example produced using an algorithm similar to that found on the Subset Sum Problem Wiki page.

Construct matrix Q, given set v = [-1, -2, 3]

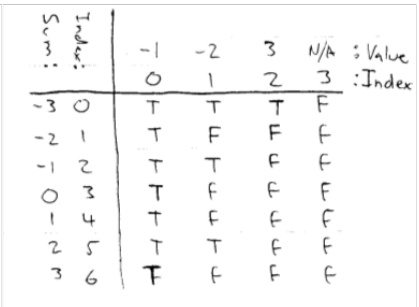
```
Q[s][n] = ( Q[s][n+1] ) || ( Q[s - v[n]][n+1] ) || ( s + v[n] == 0 )
```

Described as the exclude case, include case or self case with respect to the value described by 'n' (i.e. should the sum exclude v[n], include v[n] or does v[n] bring the sum to 0). Consult subsetsum_matrix() below for more detail.

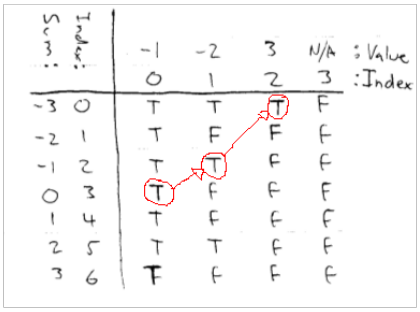
Initial Matrix



Filled Matrix



Traversed Matrix



We wrote some C code to solve this challenge based on the dynamic programming approach described above. Using C allowed us to get the performance improvements offered by both dynamic programming and native code.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

// Constants
#define LENGTH 28
#define RANGE 1170
#define SIZE (RANGE * LENGTH * 2 + 1)
#define BUFFER_SIZE 256

// Statically allocated matrix to store result
// Q is the big important matrix
// Q[all possible sum combinations][each value in set]
static unsigned char Q[SIZE][LENGTH+1];

// Given a result matrix, traverse it to find a solution of a certain length
// l: length
// v: values array
// S: solution array
// s and n: sum and index in result matrix
// b and a: negative and positive sums for matrix offset
int answer_length( int l, int *v, int *S, int s, int n, int b, int a )
{

	// no solution found within length
	if ( l <= 0 )
		return 0;

	// solution found at correct length
	if ( v[n] + s == 0 && l == 1 ) {
		S[n] = 1;
```

```
                return 1;
        }

        // traverse excluding "me"
        if ( Q[s-b][n+1] && answer_length( l, v, S, s, n+1, b, a ) )
                return 1;

        // traverse include "me"
        int temp = s + v[n];
        if ( temp >= b && temp <= a && Q[s-b+v[n]][n+1] && answer_length( l-1, v,
S, s+v[n], n+1, b, a ) ) {
                S[n] = 1;
                return 1;
        }
}

// Populate result matrix
int subsetsum_matrix( int *values, int a, int b )
{
        int s, n;
        int diff = a - b;

        // initialise to false
        for ( s = 0; s <= diff; s++ ) {
                for ( n = 0; n <= LENGTH; n++ )
                        Q[s][n] = 0;
        }

        // forgive the confusing indexing
        // the Q matrix handles all possible sum values: negative and positive
        // therefore, the most negative possible sum is actually index 0
        // thus, each sum has been offset by the most negative possible value (b)
        // E.G. s == 0, Q[s-b] is the index for sum == 0, keeping in mind b is very
negative

        // populate sum values bottom up
        for ( n = LENGTH - 1; n >= 0; --n ) {
                for ( s = b; s <= a; ++s ) {

                        // each time consider the state at sum "s" and index "n"
                        // some solutions described this as the classic "knapsack"
technique
                        // after the famous knapsack problem

                        // am "I" a solution
                        int self = values[n] + s == 0;

                        // excluding "me" is there a solution
                        int exclude = Q[s-b][n+1];
```

```
                        // including "me" is there a solution
                        int include = 0;
                        int temp = s + values[n];
                        if ( temp >= b && temp <= a )
                                include = Q[s-b+values[n]][n+1];

                        // save result
                        Q[s-b][n] = self || exclude || include;
                }
        }

        return Q[-b][0];
}

// auxiliary line reading function
ssize_t read_line( int fd, char **buf )
{
        ssize_t size = BUFFER_SIZE, length = 0;
        char c;
        *buf = malloc( sizeof( char ) * (size + 1) );

        while( read( fd, &c, 1 ) != 0 && c != '\n' )
        {
                // save character
                (*buf)[length++] = c;

                // extend buffer if necessary
                if ( length == size ) {
                        size += size;
                        *buf = realloc( *buf, sizeof( char ) * (size + 1) );
                }
        }

        // null terminate
        (*buf)[length] = '\0';

        return length;
}

int main( int argc, char *argv[] )
{
        int *values = malloc( sizeof( int ) * LENGTH );
        int *solution = malloc( sizeof( int ) * LENGTH );
        char *buf;
        int i, success;

        // connect to server
        struct addrinfo hints, *res;
        memset( &hints, 0, sizeof( struct addrinfo ) );
        hints.ai_family = AF_INET;
```

```
hints.ai_socktype = SOCK_STREAM;
getaddrinfo( "172.16.1.20", "9876", &hints, &res );
int fd = socket( res->ai_family, res->ai_socktype, res->ai_protocol );
connect( fd, res->ai_addr, res->ai_addrlen );
FILE * fp;
fp = fdopen( fd, "w" );

int num = 0;
char *tok;
char *str;

// read banner ( no error handling :O! )
read_line( fd, &buf );
printf( "%s\n", buf );
free( buf );
read_line( fd, &buf );
printf( "%s\n", buf );
free( buf );

for( num = 0; num < 10; num++ )
{

	// read round number
	read_line( fd, &buf );
	printf( "%s\n", buf );
	free( buf );

	// read length
	read_line( fd, &buf );
	printf( "%s\n", buf );
	for( str = buf; *str && *str != ':'; str++ );
	int length = atoi( ++str );
	free( buf );

	// Parse ints in line
	read_line( fd, &buf );
	printf( "%s\n", buf );
	for( str = buf; *str && *str != ':'; str++ );
	tok = strtok( ++str, " " );
	for( i = 0; tok && i < LENGTH; i++ ) {
		values[i] = atoi( tok );
		tok = strtok( NULL, " " );
	}
	free( buf );

	// calculate subset sum
	memset( solution, 0, sizeof( int ) * LENGTH );
	int a = 0; // positive sum
	int b = 0; // negative sum
	for ( i = 0; i < LENGTH; i++ )
```

```
                        if ( values[i] > 0 )
                                a += values[i];
                        else
                                b += values[i];
                success = subsetsum_matrix( values, a, b ) &&
                        answer_length( length, values, solution, 0, 0, b, a );

                // print solution
                if ( success ) {
                        for ( i = 0; i < LENGTH; ++i )
                                if ( solution[i] ) {
                                        fprintf( fp, "%d ", values[i] );
                                        printf( "%d ", values[i] );
                                }
                        fprintf( fp, "\n" );
                        printf( "\n" );
                        fflush( fp );
                } else {
                        fprintf( fp, "no solution\n" );
                        printf( "no solution\n" );
                        fflush( fp );
                }

                // read "correct"
                read_line( fd, &buf );
                printf( "%s\n", buf );
                free( buf );
        }

        // read flag
        read_line( fd, &buf );
        printf( "%s\n", buf );
        free( buf );

        // cleanup
        fclose( fp );
        close( fd );
        free( values );
        free( solution );

}
```

We compile the C source code with gcc and then our binary to get the flag for this challenge.
PanoramaSpaceBackflip582

```
#> gcc soln.c -o soln
#> ./soln
Welcome to the Fortcerts secure server. This server is protected by a challenge
response authentication method. At Fortcerts we do not believe in security by
obscurity: the response must sum to zero. Possible responses are a list of
```

```
integers separated by spaces or the string 'no solution' (because the server is
ultra-secure sometimes there may not be a solution). Generating challenge...

Round: 1
Required response length: 12
Challenge: -1136 -358 -663 27 -952 520 54 -501 17 -704 -774 -523 477 -600 -1015
-548 225 -311 -74 -990 -527 -134 -900 245 -129 -694 -700 964
520 -501 17 -523 477 225 -74 -527 245 -129 -694 964
Correct.
Round: 2
**** SNIP ****
Round: 10
Required response length: 11
Challenge: -1090 -824 -834 328 -581 319 -617 -578 391 -1121 547 429 -377 -137
-1011 -220 -444 -962 -935 -841 -504 1165 230 -1128 -1089 -392 -334 -896
391 547 429 -377 -220 -935 -504 1165 230 -392 -334
Correct.
Flag: PanoramaSpaceBackflip582
```

Mad Coding Skillz 3 - Spelunking!

Question: FortCerts are working on a breakthrough project known as project EVATAR. Using the EVATAR interface, players use neural brain circuit interferometry to control a real person trying to escape from a dangerous scenario. In this case, the scenario is a person stuck in a cave (with steps, apparently). Write a program to control your EVATAR to find the key and escape the maze. Watch your EVATAR's step though, the ceiling may be unstable. The project EVATAR access interface is located at 172.16.1.20:7788. Also don't tell anyone, it's super hush hush.

Designed Solution: This was a lot of fun to design and build >:D When you connect, you are placed in a 10 level maze, where you can move north, south, east, west, up and down. There is a door on the bottom level and a key on the one of the top 3 levels. One problem… the roof starts to collapse when you pick up the key, requiring a near optimal escape path to the door. Solution requires a combination of DFS (e.g. BackTrack) to explore the map and optimal path (e.g. A*, Dijkstra's) to move from the key to the door.

Write Up:
We wrote code that parsed the output from the server and built a graph of the maze, we used a depth first search to explore the cave, in the process finding the key and the door. Once we had found the key and the door we used Dijkstra's algorithm via pythons shortest_path module to find the shortest path between the key and the door. We then followed the shortest path to lead our EVATAR to freedom.

Before running this Python script we need to install python-pygraph on our Kali system.

```
#> apt-get install python-pygraph
```

The Python script we wrote to solve the challenge is below.

```
import numpy
import socket
from pygraph.algorithms.minmax import shortest_path
from pygraph.classes.graph import graph
from pygraph.classes.digraph import digraph

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(('172.16.1.20', 7788))

class Node:
    x = 0
    y = 0
    z = 0
    wall = False
    visited = False
```

```
    def __init__(self, x, y, z):
        self.x, self.y, self.z = x, y, z

    def get_id(self):
        return "%d:%d:%d" % (self.x,self.y,self.z)

class Searcher:

    current_node = None
    graph = graph()

    def __init__(self):
        start_node = Node(0,0,0)
        self.current_node = start_node
        self.graph.add_node(start_node.get_id(), start_node)

    def move(self, dest_node):
        if self.current_node == dest_node:
            return True

        m_x = dest_node.x - self.current_node.x
        m_y = dest_node.y - self.current_node.y
        m_z = dest_node.z - self.current_node.z

        print m_x, m_y, m_z

        if m_x == 1:
            command = 'east'
        elif m_x == -1:
            command = 'west'
        elif m_y == 1:
            command = 'north'
        elif m_y == -1:
            command = 'south'
        elif m_z == 1:
            command = 'up'
        elif m_z == -1:
            command = 'down'
        else:
            raise Exception('Invalid dest node')

        print 'Sending command %s' % command
        s.send('%s\n' % command)

        while True:
            d = s.recv(1024)
            print d

            if d.find('There is a key here') != -1:
```

```
                self.key = self.current_node

            if d.find('There is a locked door') != -1:
                self.door = self.current_node

            if d.find('You moved') != -1:
                if not self.graph.has_edge((self.current_node.get_id(),
dest_node.get_id())):
                    self.graph.add_edge((self.current_node.get_id(),
dest_node.get_id()))
                self.current_node = dest_node
                return True
            elif d.find('There is a wall') != -1:
                dest_node.wall = True
                return False
            elif d.find('No stairs') != -1:
                return False

    def find_or_create(self, x, y, z):
        n = Node(x, y, z)

        if not self.graph.has_node(n.get_id()):
            print "node not found: %s" % n.get_id()
            self.graph.add_node(n.get_id(), n)
            return n
        else:
            print "node found: %s" % n.get_id()
            return self.graph.node_attr[n.get_id()]

    def search(self, dest_node):
        if dest_node.wall:
            print 'wall at %s' % (dest_node.get_id())
            return False
        elif dest_node.visited:
            print 'visited at %s' % (dest_node.get_id())
            return False

        if not self.move(dest_node):
            return False

        print 'visiting %s' % (dest_node.get_id())

        # mark as visited
        dest_node.visited = True

        # explore neighbors clockwise starting by the one on the right
        search_nodes = [
            self.find_or_create(dest_node.x+1, dest_node.y, dest_node.z),
            self.find_or_create(dest_node.x, dest_node.y-1, dest_node.z),
            self.find_or_create(dest_node.x-1, dest_node.y, dest_node.z),
```

```
            self.find_or_create(dest_node.x, dest_node.y+1, dest_node.z),
            self.find_or_create(dest_node.x, dest_node.y, dest_node.z+1),
            self.find_or_create(dest_node.x, dest_node.y, dest_node.z-1)
        ]

        for node in search_nodes:
            print "searching node: %s" % node.get_id()
            if self.search(node):
                return True
            self.move(dest_node)
        return False

    def fill_edges(self):
        for i in self.graph.nodes():
            dest_node = self.graph.node_attr[i]
            if dest_node.wall: continue
            if not dest_node.visited: continue
            search_nodes = [
                self.find_or_create(dest_node.x+1, dest_node.y, dest_node.z),
                self.find_or_create(dest_node.x, dest_node.y-1, dest_node.z),
                self.find_or_create(dest_node.x-1, dest_node.y, dest_node.z),
                self.find_or_create(dest_node.x, dest_node.y+1, dest_node.z)
            ]
            for j in search_nodes:
                if j.wall: continue
                if not j.visited: continue
                if not self.graph.has_edge((j.get_id(), dest_node.get_id())):
                    self.graph.add_edge((j.get_id(), dest_node.get_id()))
                    print 'missing edge'

    def move_to_pos(self, dest_node):
        st,weights = shortest_path(self.graph, self.current_node.get_id())
        gst = digraph()
        gst.add_spanning_tree(st)
        print weights[dest_node.get_id()]

        path = []
        while True:
            path.append(dest_node)
            if dest_node == self.current_node: break
            dest_node = self.graph.node_attr[gst.incidents(dest_node.get_id())[0]]

        path.reverse()
        for i in path:
            print i.get_id()
            self.move(i)


cl = Searcher()
cl.search(cl.current_node)
```

```
cl.move_to_pos(cl.key)
cl.fill_edges()
s.send('pickup')
cl.move_to_pos(cl.door)
s.send('escape')
print s.recv(1024)
print s.recv(1024)
```

We run the script and after a small wait and plenty of output we receive the flag for this challenge. **TroubleStudentsRealize972**

```
#> python soln.py
visiting 0:0:0
node not found: 1:0:0
node not found: 0:-1:0
node not found: -1:0:0
node not found: 0:1:0
node not found: 0:0:1
node not found: 0:0:-1
searching node: 1:0:0
1 0 0
Sending command east
Please wait while the map loads…

Map loaded.

You moved east.

visiting 1:0:0
node not found: 2:0:0
node not found: 1:-1:0
**** SNIP ****
59:10:-5
0 1 0
Sending command north
Get out of here!

You moved north.
Get out of here!
There is a locked door here.

YOU ESCAPED!

Key: TroubleStudentsRealize972
```