Solving the T2 '13 challenge – by Ludvig Strigeus

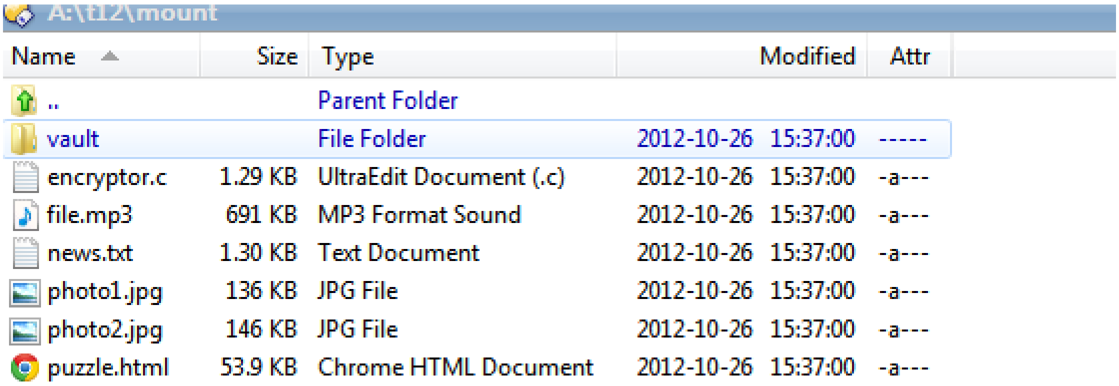
The challenge is in the form of a 256MB file called `apt.img`. It's a disk image of a USB drive. Using the `file` command verifies that this is the case:

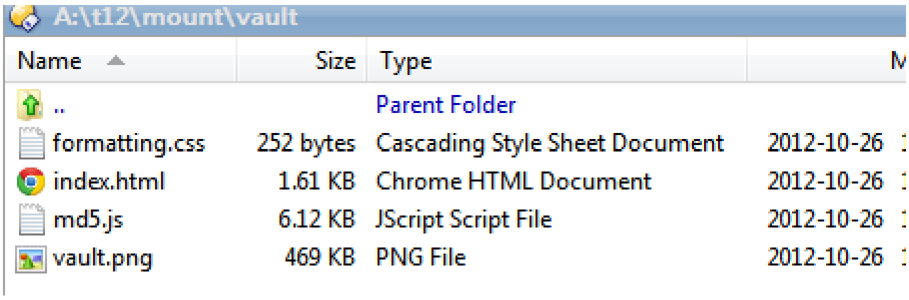
```
$ file apt.img
apt.img: x86 boot sector; partition 1: ID=0xc, active, starthead 32, startsector 2048, 497952
sectors, code offset 0x7b
```

I mount the disk image on my FreeBSD system:

```
$ mdconfig -a -t vnode -f apt.img -u 0
$ mount_msdosfs /dev/md0s1 mount
```

The USB drive contains this folder structure:

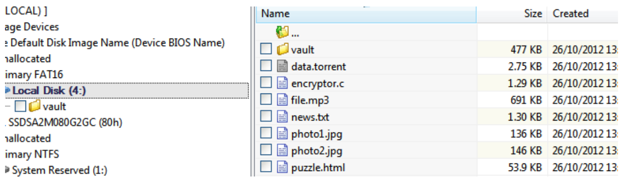




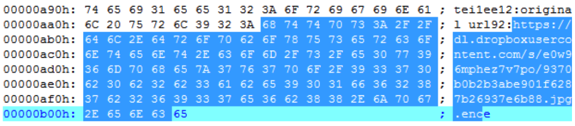
The task is to find a bunch of other files hidden or referenced inside of those files, and we need to submit the MD5 hash for each found file to the challenge web site.

The deleted torrent file

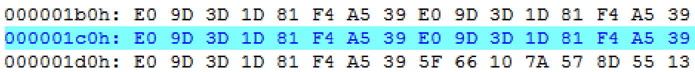
I suspected that the USB partition might have traces of deleted files. Deleting files doesn't normally overwrite the file with zeros, it merely writes in the directory entry that the file has been deleted. I used a tool called Active UNDELETE[1].



Here we see a `data.torrent` file, and I tell the undeleter to recover this file. I tried to open it with a program I made in the past, named µTorrent[2], but unfortunately there were no active seeds and peers, so the file could not be downloaded.

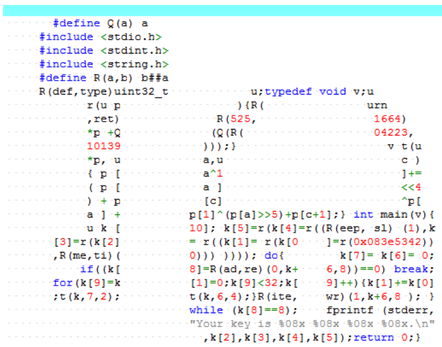


A peek inside of the .torrent file reveals a hidden URL. I successfully download this file, but it's not a valid JPEG file. The file extension of the URL hints that it's encrypted:
https://dl.dropboxusercontent.com/s/e0w96mphez7v7po/9370b0b2b3abe901f6287b26937e6b88.jpg.enc



Early in the file are some repeating sequences of 8 bytes. This suggests it has been encrypted with an encryption tool with an 8 byte blocksize. This early in the file is likely to be the JPEG EXIF[3] header. I know from experience that this header often contains runs of zeros, so my guess is that this is what `00 00 00 00 00 00 00 00` looks like when encrypted.

On the USB disk is a file named `encryptor.c`. It has been obfuscated into a funny lock and key. I restructure the code to make it readable. I discover that the encryption key is generated from time(NULL), which returns the number of seconds since 1970 as a standard UNIX timestamp. This means that we can guess when the tool was run to predict the key!
I restructure and clean up the loop of the encryptor to convert it into a decryptor. This code runs for each 8 byte block and decrypts the 8 bytes in k[0], k[1]. A,B,C,D are derived from the timestamp.



```
i = 31;
j = some_constant * 32;
do {
  k[1] -= (k[0] << 4) + C ^ k[0] + j ^ (k[0]>>5) + D;
  k[0] -= (k[1] << 4) + A ^ k[1] + j ^ (k[1]>>5) + B;
  j -= some_constant;
} while (i--);
```

I attempt a known plaintext attack[4]. My hypothesis is that `E0 9D 3D 1D 81 F4 A5 39` decrypts to all zeros. It's just a matter of testing keys until I find a match. I quickly loop through all seconds between 2011 and 2013, and find a match at 1351247820. `time.ctime(1351247820)` in Python[5] reveals that this is "Fri Oct 26 12:37:00 2012" which is pretty close to the modification date of encryptor.c, so I know I'm right. I decrypt the file with the key and get an image.

The image's MD5 hash is: **476d9247463dd91488fbd0d123e04ac1**

[1] http://www.active-undelete.com/
[2] http://www.utorrent.com/
[3] http://en.wikipedia.org/wiki/Exchangeable_image_file_format
[4] http://en.wikipedia.org/wiki/Known-plaintext_attack
[5] http://www.python.org/

The encrypted zip file

On the USB disk is a file called `news.txt`. I recognize it as a uuencoded[6] message body.

```
X-Received: by 10.224.42.141 with SMTP id s13mr5156612qae.3.1371372012214; Thu, 25 Oct 2012 15:40:12 -0700 (PDT)
X-Received: by 10.49.63.195 with SMTP id i3mr213628qes.29.1371372012130; Thu, 25 Oct 2012 15:40:12 -0700 (PDT)
Path: y6ni2993qax.0!nntp.google.com!j2no362507qak.0!postnews.google.com!glegroupsg2000goo.googlegroups.com!not-for-mail
Newsgroups: alt.test
Date: Thu, 25 Oct 2012 15:40:12 -0700 (PDT)
Complaints-To: groups...@google.com
Injection-Info: glegroupsg2000goo.googlegroups.com; posting-host=88.112.19.187; posting-account=mFzRFgoAAADTvh2hKq2sr1or8-sKyyxk
NNTP-Posting-Host: 88.112.19.187
User-Agent: G2/1.0
MIME-Version: 1.0
Message-ID: <49e33a53-4a52-4e95-9e61-0ec37ecf5763@googlegroups.com>
Subject: PWNED!
From: "albert.p.tetrick@hushmail.com" <albert.p.tetrick@hushmail.com>
Injection-Date: Thu, 25 Oct 2012 22:40:12 +0000
Content-Type: text/plain; charset=ISO-8859-1

begin 664 msg.txt
M9V]T(&UE;2!D=6UP(&]F(%0G<R!B;W@@0"!:9710<BX@:2!C86X@:&%S('!W
M(&AA<V@@9'5M<#\@=VEN('!W(#T]('II<"!P=S\_/R!V;VQA=&EL:71Y(&9T
M=R$*"FAT=''!S.B\O9&PN9')O<&)O>'5S97)C;VYT96YT+F-O;2]U+S(X.#4Q
M-C(P+U0O830T-&-F-C!F,3,S.#)C8C%C,C,S,S8S-S@Q,C8U,S0Y-#@X-38S
M82YZ:7`*:'1T<',Z+R]D;"YD<F]P8F]X=7-E<F-O;G1E;G0N8V]M+W4O,C@X
M-3$V,C`O5"\V-SEF.6$Y-S,W96-B-#)C8S4V83$V-F8S930X,S!E,C(U-#0X
09&8Q+GII<`H*+2T@05!4"@``
`
end
```

```
$ uudecode news.txt
$ cat msg.txt
```

```
got mem dump of T's box @ Zetor. i can has pw hash dump? win pw == zip pw??? volatility ftw!
https://dl.dropboxusercontent.com/u/28851620/T/a444cf60f13382cb1c23336378126534948856 3a.zip
https://dl.dropboxusercontent.com/u/28851620/T/679f9a9737ecb42cc56a166f3e4830e225448df1.zip

-- APT
```

The first file contains a 2GB memory dump. The second file contains `e79d2f8834910399c34192a2f1f8fc0e.jpg`, but it's been encrypted with a zip file password.

I search on Google for a tool to dump passwords from RAM memory dumps. I find a tool called **volatility**[7], nice hint there. I use a guide[8] that decribes how to use volatilty.

```
$ volatility-2.1.standalone.exe imageinfo -f T.raw
Volatile Systems Volatility Framework 2.1
Determining profile based on KDBG search...
          Suggested Profile(s) : WinXPSP2x86, WinXPSP3x86 (Instantiated with WinXPSP2x86)

$ volatility-2.1.standalone.exe hivelist -f T.raw --profile=WinXPSP2x86
Volatile Systems Volatility Framework 2.1
Virtual    Physical   Name
---------- ---------- ----
0xe14b4008 0x0f962008 \Device\HarddiskVolume1\WINDOWS\system32\config\SAM
0xe1035b60 0x0a424b60 \Device\HarddiskVolume1\WINDOWS\system32\config\system
...

$ volatility-2.1.standalone.exe hashdump -f T.raw --profile=WinXPSP2x86 -y 0xe1035b60 -s 0xe14b4008
Volatile Systems Volatility Framework 2.1
T:500:81de36ec83691f0b22d3b69d51786748:bdf8f7ed94d3358e2be2b16ae602cf20:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
HelpAssistant:1000:469c976ed23a70e0799d1f5c3b02f777:1b67841672576a7a9f46e1c55f987b99:::
SUPPORT_388945a0:1002:aad3b435b51404eeaad3b435b51404ee:1157c1e88d4918b8bd230f633937c871:::
```

Here above we see the LM and NTLM hashes for the user T. I use a web based tool named ophcrack[9] based on rainbow tables[10] that cracks LM and NTLM hashes. It reveals that the password is **t2infosec**. Now I can unzip the jpeg file with this password, just like `msg.txt` said.

```
$ md5sum.exe e79d2f8834910399c34192a2f1f8fc0e.jpg
e79d2f8834910399c34192a2f1f8fc0e *e79d2f8834910399c34192a2f1f8fc0e.jpg
```

How silly, the name of the file is the same as its hash. There was no need to do the password cracking. Was that a glitch in the challenge?

The image's MD5 hash is: **`e79d2f8834910399c34192a2f1f8fc0e`**

[6] http://en.wikipedia.org/wiki/Uuencoding
[7] https://www.volatilesystems.com/default/volatility
[8] http://cyberarms.wordpress.com/2011/11/04/memory-forensics-how-to-pull-passwords-from-a-memory-dump/
[9] http://www.objectif-securite.ch/en/ophcrack.php
[10] http://en.wikipedia.org/wiki/Rainbow_table

The damaged QR code

On the USB disk is a photo named photo2.jpg. The upper corner is missing. I see that it contains 29x29 pixels, so I rotate, crop and resize the photo down to this size to get a smaller bitmap with nothing but the QR code pixels. Unfortunately the top corner is missing. I'm unable to find a tool capable of decoding this broken code. My phone can't decode it either.

Original QR-code

Wikipedia[11] describes QR codes quite well, and they contain error correcting codes, meaning that I don't need the full code in order to recover the contents. I fill in the pixels of the blank part with the missing format, timing zone and positioning elements[12]. Those elements are redundant so I copied them from the other place in the image. The decoder probably couldn't find the code when those fields were missing.

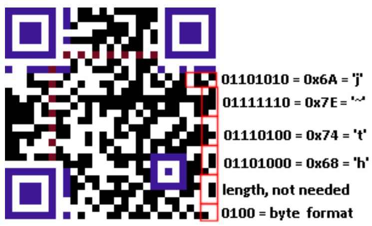
Slightly fixed QR-code

Now I have some more luck! I manage to partially decode the QR code with a web based tool[13] into the string:
`ht~j://tiny□��.com/yNS�_W�utionzo|s>omgz`

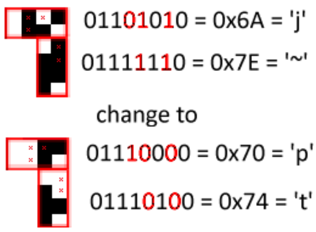
It's obvious that the string should start with http. I can use this knowledge to manually repair a few pixels and then the error correcting codes might be able to fix the rest. To be able to edit the characters in the image I need to know which masking mode is used, and I need to know what character encoding that is used.
The masking mode is because the QR-code encoder has XOR:ed the pixels with one of eight different masks[14], in order to make the pattern easier to parse. The masking mode is encoded in the 15 blue format bits, and those bits say that it's mask number 6 that has been used. I wrote a small tool in Python to XOR two images and produce an unmasked version.

The layout[15] of the 29x29 pixels QR code shows that the data bytes D1, D2, .. start in the lower right corner and go upwards. Wikipedia explains that the bottom right 2x2 pixels determine the encoding mode. Those pixels are **0100** after unmasking which means byte encoding with ASCII[16] characters. The next 8 pixels are the size, and after that we should see the ASCII values for the "ht~j" characters.

Masking image



Unmasked QR-code with decoded characters



Pixels to repair hilighted with red

Fixed QR-code with pixels flipped

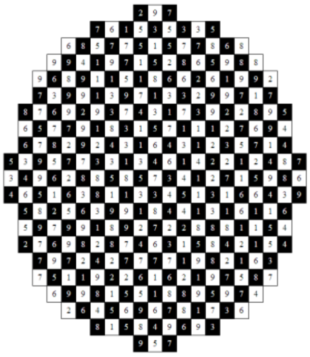
Once I know which pixels need repairing, I can just flip them in the original image. There's no need to do a separate mask step. After flipping the 5 broken pixels in the original image, I ended up with a code that successfully decodes! This means we now have enough correct pixels for the error correction to work. The contained text is `http://tinyurl.com/ihazsolutionzomgzomgz` and this URL shows another picture of a woman.

The image's MD5 hash is: `60e327e0fac73eb6fa291bff84497c2a`

[11] http://en.wikipedia.org/wiki/QR_code
[12] http://en.wikipedia.org/wiki/File:QRCode-2-Structure.png
[13] http://www.esponce.com/qr-code-decoding
[14] http://research.swtch.com/qart19.png
[15] http://en.wikipedia.org/wiki/File:QRCode-3-Layout,Encoding.png
[16] http://www.asciitable.com/

Back from the Klondike

This challenge started with a picture of an old man. I used a similar image search engine[17] on his picture to find that his name is Sam Loyd[18] and he was born in 1841. Great!



He is famous for designing puzzles, and he made a puzzle named "Back from the Klondike"[19] which looks like `puzzle.html` on the USB drive.

The puzzle instructions say:
Start from the center. Go three steps in a straight line in any one of the eight directions, north, south, east, west, northeast, northwest, southeast, or southwest. When you have gone three steps in a straight line you will reach a square with a number on it, which indicates the second day's journey, as many steps as it tells, in a straight line in any one of the eight directions. From this new point, march on again according to the number indicated, and continue on in this manner until you come upon a square with a number which will carry you just one step beyond the border, thus solving the puzzle.

Unfortunately, this puzzle's cell numbers differ from those of the original "Back from the Klondike", so I need to make a puzzle solver. I implement a breadth first search[20] in C++, that starts in the center, and jumps the right number of steps in all eight directions, and continues like this until it reaches a cell one step beyond the border, while keeping track of the visited path. One thing that greatly speeds up the search is that I don't need to revisit cells I've already visited.

Within a second, the program solves it and says:
`Found solution:  NW E SE SW NW N E SE SW NW N`

On the USB stick is a vault folder with an `index.html` that looks like the picture on the right. It wants you to key in a bunch of directions. When done, press the lock, and a URL will be visited that is constructed from the directions pressed. I key in the directions my program told me, and I get the solution picture.

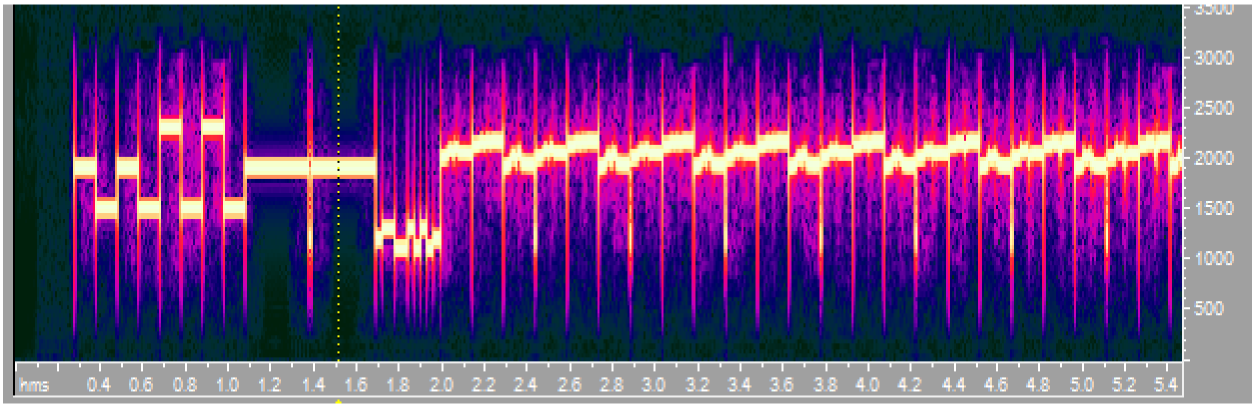
The image's MD5 hash is: **`c321553877c582edc9435f97f5bcd7e7`**

[17] http://www.tineye.com/
[18] http://en.wikipedia.org/wiki/Sam_Loyd
[19] http://en.wikipedia.org/wiki/Back_from_the_Klondike
[20] http://en.wikipedia.org/wiki/Breadth-first_search

Slow-scan television

The only remaining file now is an mp3 file called `file.mp3`. Playing it back in a music player reveals nothing, except that it sounds like a modulated signal.

I open the file with my good old CoolEdit 96 tool[21], and display it in the spectrum mode. This image shows the frequency components of the audio signal. I'm a bit clueless about what modulation this is. It doesn't sound at all like the modems used to sound back in the days, so it's probably not a modem transmission.



I am stuck for quite some time, measure the various tone frequencies by looking at the Y axis, 1900Hz, 1500Hz and 2300Hz but I don't recognize the signal. I hear small clicks every 450ms, I find that this is a phase reversal[22] used to turn off the echo cancellation[23] of analogue networks. So indeed it must be a signal transmitted over phone lines. The long flat line looks like some kind of sync signal. I search on Google for the string "1900Hz modulated signal" and it gives me a good result back on the first page:

Slow-scan television - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/Slow-scan_television
This **signal** can be fed into an SSB transmitter, which in part **modulates** the **...** It consists of a 300-millisecond leader tone at **1900 Hz**, a 10 ms break at 1200 Hz, **...**

Wikipedia[24] says:
"A calibration header is sent before the image. It consists of a 300-millisecond leader tone at 1900 Hz, a 10 ms break at 1200 Hz, another 300-millisecond leader tone at 1900 Hz, followed by a digital VIS (vertical interval signaling) code, identifying the transmission mode used."

This is exactly what I see. I see a 300ms flat line, then a interruption, and another 300ms flat tone at 1900Hz! The sound is an image!

Wikipedia links to a software named RX-SSTV[25] that can be used to decode such pictures. I install it, and play the mp3-file while the software is recording. It slowly reveals a picture with a URL pointing at an image of the lady!

The image's MD5 hash is:
4e20c6c8d8b7473a2be84b12ab837bfd

---

[21] http://www.threechords.com/Hammerhead/cool_edit_96.shtml
[22] http://tools.ietf.org/html/rfc4734
[23] http://en.wikipedia.org/wiki/Echo_canceller
[24] http://en.wikipedia.org/wiki/Slow-scan_television
[25] http://users.belgacom.net/hamradio/rxsstv.htm

Appendix A – Source code to encryptor key generator

```
#include <stdio.h>
#include <stdint.h>
#include <string.h>

uint32_t r(uint32_t p){
 return 1664525* p + 1013904223;
}

int decrypt(uint32_t key){
 uint32_t k[2];
 uint32_t i,j,A,B,C,D;
 uint32_t num, some_constant;
 unsigned char ciphertext[8] = {0xe0, 0x9d, 0x3d, 0x1d, 0x81, 0xf4, 0xa5, 0x39};
 unsigned char plaintext[8] = {0};

 C=r((B=r(A = r((j= r(some_constant=r(0x083e5342)), key)))));
 D=r(C);

 memcpy(k, ciphertext, 8);
 i = 31;
 j = some_constant * 32;
 do {
  k[1] -= (k[0] << 4) + C ^ k[0] + j ^ (k[0]>>5) + D;
  k[0] -= (k[1] << 4) + A ^ k[1] + j ^ (k[1]>>5) + B;
  j -= some_constant;
 } while (i--);
 if (memcmp(k, plaintext, 8) == 0) printf("The time(NULL) is %d\n", key);
}

int main(void) {
  int i;
  for(i = 1288103820; i  < 1382798220; i++)
    decrypt(i);
  return 0;
}
```

Appendix B – Source code to decryptor

```
#include <stdio.h>
#include <stdint.h>
#include <string.h>

uint32_t r(uint32_t p){
 return 1664525* p + 1013904223;
}

void decrypt(uint32_t key){
 uint32_t k[2];
 uint32_t i,j,A,B,C,D;
 uint32_t num, some_constant;
 C=r((B=r(A = r((j= r(some_constant=r(0x083e5342)), key)))));
 D=r(C);
 do{
   k[1] = k[0] = 0;
   num = read(0, k, 8);
   if (num == 0) break;
   i = 31;
   j = some_constant * 32;
   do {
    k[1] -= (k[0] << 4) + C ^ k[0] + j ^ (k[0]>>5) + D;
    k[0] -= (k[1] << 4) + A ^ k[1] + j ^ (k[1]>>5) + B;
    j -= some_constant;
   } while (i--);
   write(1, k, 8);
 } while (num==8);
}

int main(void) {
  decrypt(1351247820);
  return 0;
}
```

Appendix C – Source code to puzzle solver

```
#include <stdio.h>
#include <deque>
#include <string>

char *map[] = {
"                  2 9 7                     ",
"            7 6 1 5 3 5 3 3 5               ",
"        6 8 5 7 7 5 1 5 7 7 8 6 8           ",
"      9 9 4 1 9 7 1 5 2 8 6 5 9 8 8         ",
"    9 6 8 9 1 1 5 1 8 6 6 2 6 1 9 9 2       ",
"    7 3 9 9 1 3 9 7 1 3 3 2 9 9 7 1 7       ",
"  8 7 6 9 2 9 3 7 4 3 1 7 3 9 2 2 8 9 5     ",
"  6 5 7 7 9 1 8 3 1 5 7 1 1 1 2 7 6 9 4     ",
"  6 7 8 2 9 2 4 3 1 6 4 3 1 2 3 5 7 1 4     ",
"5 3 9 5 7 7 3 3 1 3 4 6 1 4 2 2 1 2 4 8 7   ",
"3 4 9 6 2 8 8 5 8 5 7 3 4 1 2 7 1 5 9 8 6   ",
"4 6 5 1 6 3 8 1 1 3 3 4 5 1 3 1 6 6 4 3 9   ",
"  5 8 2 5 6 3 9 9 1 8 4 4 1 3 1 6 1 1 6     ",
"  5 9 7 9 9 1 8 9 2 7 2 2 8 8 8 1 1 5 4     ",
"  2 7 6 9 8 2 8 7 4 6 3 1 5 8 4 2 1 5 4     ",
"    7 9 7 2 4 2 7 7 7 7 1 9 8 2 1 6 3       ",
"    7 5 1 1 9 2 2 6 1 6 2 1 9 7 5 8 7       ",
"      6 9 9 8 1 5 5 1 8 8 9 5 9 7 4         ",
"        2 6 4 5 6 9 6 7 8 1 7 3 6           ",
"            8 1 5 8 4 9 6 9 3               ",
"                  9 5 7                     ",
};

int Get(int x, int y) {
  return (x < 0 || y < 0 || x >= 22 || y >= 21) ? ' ' : map[y][x*2];
}

struct S {
  int x,y;
  std::string d;
  S(int x, int y, const std::string &d) : x(x), y(y), d(d) {}
};

int dirs[8][2] = {{1,0},{-1,0},{0,1},{0,-1},{1,-1},{1,1},{-1,-1},{-1,1}};
char *dirtxt[] = {" E", " W", " S", " N", " NE", " SE", " NW", " SW"};
bool visited[32][32];

int main(int argc, char* argv[]) {
  std::deque<S> deq;
  deq.push_back(S(10, 10, ""));
  while (!deq.empty()) {
    S s = deq.front();
    deq.pop_front();
    int d = Get(s.x,s.y);
    if (visited[s.y][s.x]) continue;
    d -= '0';
    visited[s.y][s.x] = true;
    for(int i = 0; i < 8; i++) {
      for(int j = 1; j <= d; j++) {
        if (Get(s.x + dirs[i][0] * j, s.y + dirs[i][1] * j) == ' ') {
          if (j == d) printf("Found solution: %s%s\n", s.d.c_str(), dirtxt[i]);
          goto NEXT;
        }
      }
      deq.push_back(S(s.x + dirs[i][0] * d, s.y + dirs[i][1] * d, s.d + dirtxt[i]));
NEXT:;
    }
  }
  return 0;
}
```