Design of Embedded System-Based CAPTCHA in C

Bo Qu
School of Mathematics and Information Technology
Nanjing Xiaozhuang University
Nanjing, China
e-mail: Mr.QuBo@126.com

Zhaozhi Wu
School of Mathematics and Information Technology
Nanjing Xiaozhuang University
Nanjing, China
e-mail: wzz5958@126.com

***Abstract*—This paper describes the technical details of design and implementation of embedded system-based CAPTCHA (verification code) in C, including a brief analysis of techniques for implementing CAPTCHA, and a simple and practical CAPTCHA system designed by the author of this paper, which can be used not only in embedded systems but also ordinary Web based MIS or control systems, without the support of any dedicated graphics library. An example is provided to show the practical usage and effect of this CAPTCHA system on an embedded application demo system.**

Keywords-CAPTCHA; secure login; embedded system; bitmap image; C programming

I. INTRODUCTION

With the rapidly technical developments of networks and embedded systems, the B/S (Browser/Server) mode [1] has almost been the standard one for users to communicate with remote embedded systems. The basic form of this is by using a browser in a client computer to login the embedded system then access or control the remote system. Obviously such a login system must use the username/password verification mechanism to prevent invalid users getting into the system. In the early stage of networks it is really useful, whereas some malicious software, such as password detectors, etc., has been developed now to probe the login systems automatically [2].

To deal with these malicious programs, a new technique emerged as time requires, named CAPTCHA [3]. The term "CAPTCHA" is an acronym based on the word "capture" and standing for "Completely Automated Public Turing test to tell Computers and Humans Apart", the purpose of which originally is to differentiate between human and automated programs by setting some task that is on one hand easy for most humans to perform while on other hand is more difficult and time-consuming for current bots to complete [6].

For popularly used Web systems, some dedicated graphics libraries such as GD are installed on the Web server to support the requirements of such systems, including providing graphical functions to form CAPTCHA images for various Web based MIS or control systems, most of which are designed with a kind of Web authoring languages such as PHP, ASP, JSP, Perl, etc. which are obviously not suitable for designing ordinary embedded system programs.

Although graphics libraries are powerful but the sizes of them are also too large and requiring lots of system resources to run. That means they are not suitable for most embedded systems with little scale of hardware and software resources. The common ways to implement CAPTCHA systems for embedded systems without dedicated graphics libraries are to design the system with C language independent of any graphics library. The CAPTCHA system described in this paper is just such a one designed by the author of this paper.

The contributions of this CAPTCHA system are as following:

- Programming in C. Regarding the peculiarity of embedded systems, the C language is the best one for embedded system programs design [5]. The program can implement a practical CAPTCHA system which can surely meet the needs of secure login for embedded application systems, and generate effective CAPTCHA images without the supporting of any graphics library.
- Some useful functions for CAPTCHA are realized. A CAPTCHA image consisting of four to six numeric digits, each of which can be distorted horizontally, stretched vertically, and placed in different positions, of course all randomly.
- The system is simple and practical. The format of the CAPTCHA image is bitmap with 16 colors and the image size of it is 96×32 (96 columns and 32 rows of pixels). That means the file size of it is also small enough. Additionally, the entire program is elaborately designed so that its code count is only less than 200 lines. It can be used as a CGI program in an embedded login system.

This paper describes the technical details of designing and implementing this CAPTCHA system and show the usage and effect of the system by providing a simple example on an embedded application demo system.

II. DESIGN AND IMPLEMENTATION OF CAPTCHA

A. Overview of CAPACHA Systems

One of the commonly usages of a CAPTCHA system is to stop bots and other automated programs from signing up a Web based system automatically.

In order to achieve this purpose, a CAPTCHA generally features an image file of slightly distorted alphanumeric characters which can usually be read without too much difficulty by a human. Although an automated program is able to recognize that the content contains an image, however it has no idea what the image is. Fig. 1 shows some typical examples of CAPTCHA images [4].

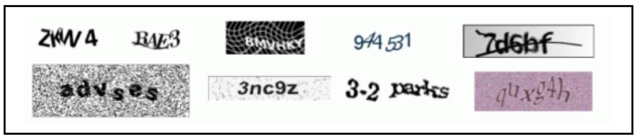


Figure 1. Some examples of CAPTCHA images

B. *Bitmap Format*

In order to save images, various image file formats are developed. One of them is called bitmap which is the simplest format corresponding to the pixels consisting of the image.

The bitmap file format, saved as .bmp as the postfix, is the standard for Windows files. The image itself can either contain pointers to entries in a color table or literal RGB values. A typical bitmap file consists of four segments [7]:

File header, information header, color palette, image data.

File header contains some information about the bitmap file (about the file, not about the bitmap image itself).

Information header contains information about the bitmap image such as size, colors, etc.

Color palette contains a color table which consists of an array to define the palette of the image.

Image data consist of pixels of the image, of which the format is specified by the above information header structure.

C. *Pixel Array of Numeric Digits*

Considering the identification efficiency by human, especially for the distorted image, only the ten numeric digit characters are used to form the image in the CAPTCHA system described in this paper.

The original bitmap for each numeric character (0, 1, …, 9) is designed as a 8×12 (8 columns by 12 rows) array. For example, the bitmap array of digit 0 is:

{0x3c,0x42,0x42,0x42,0x42,0x42,0x42,0x42,0x42,0x42, 0x42,0x3c}.

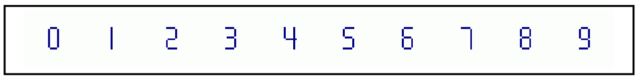


Figure 2. The original numeric digits image

The original digits image is shown in Fig. 2. The purpose of using such a “thin shaped” font is making the identification effect not too difficult as well as providing less information to the malicious automated programs.

Obviously, one pixel row can be stored in a byte, therefore the bitmap array of the 10 digits can be stored in a character array as following in C:

unsigned char digits[10][12] = { … };

D. *Color palette*

The CAPTCHA system described in this paper uses the bitmap format of 16 colors. According to the specification of bitmap format, it needs a color palette. The purpose of the color palette in indexed color bitmaps is to inform the application about the actual color that each of these index values corresponds to. It is a block of bytes (a table) listing the colors used by the image. Each pixel in an indexed color image is described by a number of bits (1, 4, or 8) which is an index of a single color described by this table. The color palette occurs in the bitmap file directly after the information header.

For the palette of 16 colors, each entry in the color table occupies 4 bytes, in the order Blue, Green, Red, and 0x00. That means the size of entire table is 64 bytes. According to HTML specification, the color names in a 16 color palette scheme are as following, sequentially:

0:black, 1:maroon, 2:green, 3:olive,
4:navy, 5:purple, 6:teal, 7:gray,
8:silver, 9:red, 10:lime, 11:yellow,
12:blue, 13:fuchsia, 14:aqua, 15:white.

The corresponding values are defined in C as following:

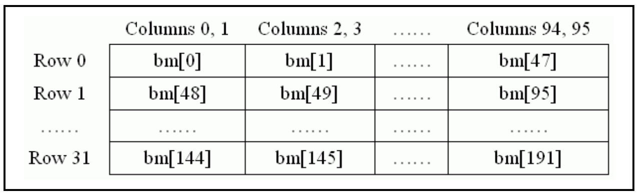
```
unsigned char rgb[64] = {
0,0,0,0, 0,0,128,0, 0,128,0,0, 0,128,128,0,
128,0,0,0, 128,0,128,0, 128,128,0,0, 128,128,128,0,
192,192,192,0, 0,0,225,0, 0,255,0,0, 0,255,255,0,
255,0,0,0, 255,0,255,0, 255,255,0,0, 255,255,255,0
};
```

E. *Pixel Array of the Bitmap Image*

For a 16 colors bitmap image, each pixel needs 4 bits to be represented therefore each byte can store 2 pixels. As mentioned above, the system described in this paper set the image size as 96 columns by 32 rows, that means the memory space needs for each row is 48 bytes. For the purpose of convenience, therefore, the array to store the image is designed as following in C program:

unsigned char bm[48 * 32];

which means there are totally 32 rows and each row 48×2 columns of pixels, as shows in Fig 3.



	Columns 0, 1	Columns 2, 3	……	Columns 94, 95
Row 0	bm[0]	bm[1]	……	bm[47]
Row 1	bm[48]	bm[49]	……	bm[95]
……	……	……	……	……
Row 31	bm[144]	bm[145]	……	bm[191]

Figure 3. Memory allocation of the pixels

In such a way to store the pixels, it will be more convenient to manipulate the entire bitmap image such as adding some background interference colors, patterns, etc.

F. *Distortions of the Digit Pixels*

In order to increase the identification difficulties for automatic programs, the digit image in the CAPTCHA should be distorted in some way.

There are various algorithms to do so and a relatively simple one is used here. In this way, the pixels of one numeric digit are manipulated with two different methods:

- In horizontal direction, an entire row of the pixels may be shifted to left or right one bit each time, or even no change, randomly determined by a random number which can be 1, 0, or -1 corresponding to left, none and right respectively. That means the digit image may be distorted in horizontal direction.

- In vertical direction, an entire row of the pixels may be duplicated to a new row bellow the current row also determined by a random number which can be 1 or 0 corresponding to duplication or none. By this way, the digit image may be stretched in vertical direction.

Combining the two methods together, the image of each numeric digit can be distorted as well as stretched randomly, as shown in Fig. 4.

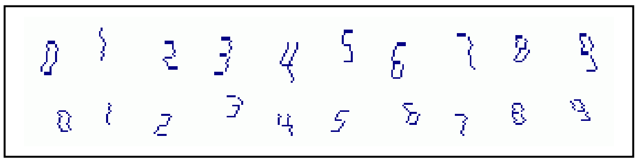


Figure 4. The distorted digit images

G. *Layout of the Digit Images*

To increase the identification difficulty further, not only the shapes of the digit images themselves are distorted, but also their positions placed in the entire CAPTCHA image are vary randomly in a predetermined range. That means, for the fixed size of the entire image columns, the nearer two digits are, the more digits might be contained. Therefore, in practice, a CAPTCHA image can contain four to six digits randomly, as shown in Fig. 5.

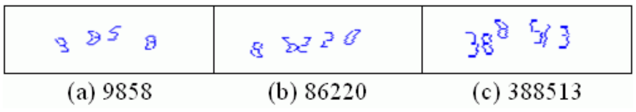


Figure 5. Various digits can be contained

The image in Fig. 5 (a) is an example of 4 digits, in which the distances between 8 and 5 is nearer while others are relatively not so near. In second image of Fig. 5, there are five digits in which 6 and the first 2 is obviously nearer. The last one in the figure shows the case of 6 digits in which the distances between first 3 and 8, 5 and 1 are nearer. For humans such images are not very difficult to be identified, however, it will significantly increase the difficulties of identification for malicious automated programs.

H. *Implementation in C*

The core function is fill_bmp() which manipulates the pixel array of one digit to fill the bitmap image space. The key code is shown in Fig. 7, in which variables k and m represent the original column and row positions of the digit in the entire image respectively as shown in Fig. 6.

The first parameter, str, represents the pixel array of the digit, and the second, bmp, represents the address in the bitmap image array at which the digits distorted image to be filled.

Note that the definition of the second parameter. It is defined as a two dimensional character array bmp[ ][3 * 16], with which the function can access the bitmap image array easily although there are only 8 columns of pixels and may use at most only 16 bytes per row for each digit.

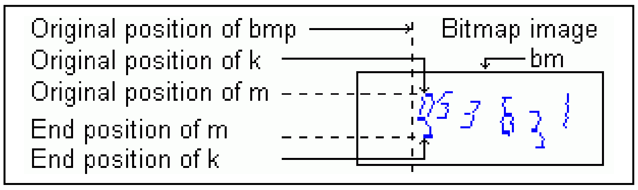


Figure 6. Fill bitmap image with a digit

```
void fill_bmp(char *str, unsigned char bmp[][3 * 16])
{
   int i, j, k = 6 + (rand() % 4);
   int m = (rand() % 12), n;

   unsigned int bit[12], st = 1;
   if ((n = (bm + 3 * 16) - &bmp[0][0]) > 16) n = 16;
   else k = n / 2;
   for (i = 0; i < 12; i ++) {
      bit[i] = str[i];
      k += (rand() % 3) - 1;
      if (k < 0) k = 0;
      else if (k > n - 1) k = n - 1;
      bit[i] <<= k;
      st = 1;
      for (j = 0; j < n; j ++) {
         if (bit[i] & st)
         bmp[m][j]=(bmp[m][j] & 0x0f) | (color << 4);
         st <<= 1;
         if (bit[i] & st)
            bmp[m][j]=(bmp[m][j] & 0xf0) | color;
         st <<= 1;
      }
      if (m < 31) m ++;
      if (m < 31 && rand() % 2) {
         for (j = 0; j < 16; j ++)
            bmp[m][j] |= bmp[m-1][j];
         m ++;
      }
   }
}
```

Figure 7. Code for function fill_bmp()

As mentioned previously, character array bm is used to store the entire bitmap image. The variable n in Fig. 7 represents the maximum width of the current manipulated digit in the bitmap image. Normally its value is 16 (32 pixels each row) while it may be less than that when the digit is the last one in the bitmap image.

Although the bitmap image is of 16 colors while only one color is used as the foreground color and one is background. The value of the foreground color, i.e. the digit color in the image, is defined as 12 and that of the background is 0. According to the definition of the color palette mentioned previously, the two colors are blue and black respectively. By modifying the values of the first four bytes in color palette, the real background color can be changed, for example, as 255, 255, 255, and 0, which make the background color becomes white.

III. APPLICATION OF THE CAPTCHA

To use the CAPTCHA in the procedures of Web login, the bitmap image generated as mentioned above must be as an image file to be sent to users' browser. The function send_bmp() accomplishes such a task. The key code of the function is shown in Fig. 8.

```
unsigned char itoa64[] =  /* 0 ... 63 => ascii - 64 */
"./0123456789ABCDEFGHIJKLMNOPQRSTUVWXY
Zabcdefghijklmnopqrstuvwxyz";

void send_bmp(char *str)
{
  char salt[2];
  int i, j, k;

  salt[0] = itoa64[time(NULL) & 0x3f];
  salt[1] = itoa64[getpid() & 0x3f];
  printf("Status:200\n");
  printf("Set-Cookie:vcode=%s;expires=12-31-2038\n",
  crypt(str, salt));
  printf ("Content-type: image/bmp\n\n");
  fwrite(bmflag,2,1,stdout);
  fwrite(&bfh,sizeof(bfh),1,stdout);
  fwrite(&bih,sizeof(bih),1,stdout);
  fwrite(rgb,sizeof(rgb),1,stdout);
  for (i = 0; i < 32; i ++)
    for (j = 0; j < CODELEN * 16; j ++)
  putc(bm[(32 - i - 1) * 3 * 16 + j], stdout);
}
```

Figure 8. Key code of function send_bmp()

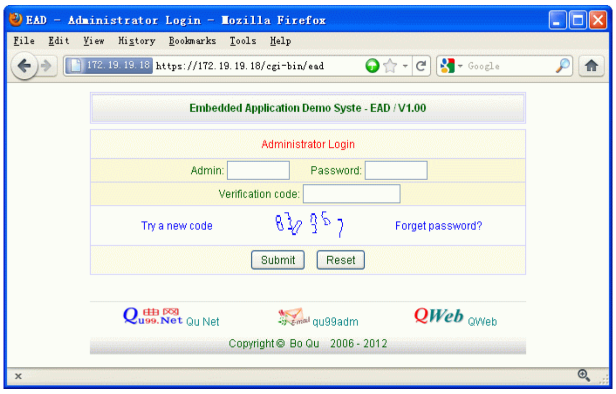
The function generates a Web page to be sent to users' browser. The Web page consists of two parts: message header and message body.

In message header, a cookie corresponding to the CAPTCHA string is sent to the browser. The cookie name is named vcode and the value of it is an encoded string by function crypt(), for which the salt is generated by the character array itoa64[ ] with the current time and the process's ID number as the indices. The content type of the page file is defined as "image/bmp". The double "\n" defines the empty line between the message header and body.

In message body, the bitmap image is as a bitmap image file sent to the browser. It first sends the 2 bytes of bitmap file flag "BM", then the file header, information header and color palette. The data of the bitmap image come last. It is worthy to note that the sequence of storing a bitmap image into the file is from bottom line up to the top.

Fig. 9 gives an example by using the CAPTCHA system in an embedded application demo system. The super-link "Try a new code" can be used to refresh the CAPTCHA image without reloading the entire page.

Confined to the length of the thesis, some of the technical details are omitted such as embedding the CAPTCHA into a Web page, further improvements of the algorithms to generate CAPTCAH images, etc.


EAD – Administrator Login – Mozilla Firefox

https://172.19.19.18/cgi-bin/ead

Embedded Application Demo Syste - EAD / V1.00

Administrator Login

Admin:

Password:

Verification code:

Try a new code

Forget password?

Copyright © Bo Qu 2006 - 2012

Figure 9. An example of CAPTCHA in an embedded system

IV. CONCLUSION

As everyone knows, there are many differences between general-purpose and embedded systems. How to make Web based embedded systems more efficient and more secure has become an important issue. The embedded system-based CAPTCHA system described in this paper is a good example on the research. It can be used not only in embedded systems but also ordinary Web based MIS or control systems. It is undoubtedly that the research in the field has great significance therefore is worthy of strengthening further.

REFERENCES

[1] A. S. Tanenbaum. "Computer Networks, Fourth Edition", Prentice Hall, inc., 2008.

[2] D. E. Comer. "Computer Networks and Internet with Internet Applications, Fourth Edition", Prentice Hall, inc., 2004.

[3] G. Mori, J. Malik. "Recognizing objects in adversarial clutter: Breaking a visual CAPTCHA", Proceedings of the 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, volume 1, pp. 134-141, 2003.

[4] L. V. Ahn, M. B. and J. Langford. "Telling Humans and Computers Apart Automatically", Communications of the ACM, volume 47, pp. 57-60, 2004.

[5] M. J. Rochkind. "Advanced UNIX Programming, Second Edition", Addison-Wesley, 2006.

[6] CAPTCHA Web Page: http://www.captcha.net, 2011.

[7] Bitmap Web Page: http://en.wikipedia.org/wiki/bitmap, 2011