Enkel beskrivning och implementering av kognitivt verifikationsschema

HENRIK KRISTIANSSON
och MARTIN ALDRIN

KTH Datavetenskap
och kommunikation

Examensarbete
Stockholm, Sverige 2010

Enkel beskrivning och implementering av kognitivt verifikationsschema

H E N R I K K R I S T I A N S S O N
o c h M A R T I N A L D R I N

Examensarbete i datalogi om 15 högskolepoäng
vid Programmet för datateknik
Kungliga Tekniska Högskolan år 2010
Handledare på CSC var Mikael Goldmann
Examinator var Mads Dam

URL: www.csc.kth.se/utbildning/kandidatexjobb/datateknik/2010/kristiansson_henrik_OCH_aldrin_martin_K10074.pdf

Kungliga tekniska högskolan
Skolan för datavetenskap och kommunikation

KTH CSC
100 44 Stockholm

URL: www.kth.se/csc

Sammanfattning

Den här rapporten behandlar kognitivt verifikationsschema, hur man kan implementera detta och hur det fungerar. Den visar även hur man kan använda detta och vilka svagheter som det har. Rapporten bygger på en rapport av Daphna Weisenhall (2006) där författaren presenterar ett schema som använder sig av kognitiv verifikation. Den använder sig av ett enkelt lösenord där användaren måste komma ihåg några bilder ur ett större antal. Detta görs genom att följa algoritmen genom matrisen och returnera en siffra vid slutet av stigen.
Vi bygger också vårt arbete på en rapport skriven av Phillip Golle och David Wagner (2007) för IEEE symposium så är det möjligt att med en satisfierbarhetslösare knäcka systemet med bara 60 lyckade utmaningar. Även om detta schema i nuläget har några allvarliga brister så kan det kanske vara användbart som en extratjänst för att logga in på en osäker plats.

Abstract

This report is about cognitive authentication schemes, how it is to be implemented and how it works. It shows how to use it and the weaknesses that comes with it. The reports mostly based on a report by Daphna Weisenhall (2006) where they present a scheme that uses cognitive verification. It uses a simple password where the user has to remember a subset of pictures as a password. This is done by following the algorithm through the matrix and returns an integer at the end of the path.
We also build our report on the report written by Phillip Golle and David Wagner (2007) for the IEEE symposium, that with a SAT solver it's possible to breach this scheme from only 60 successful answers to the challenges. Even if the scheme has some fatal flaws this scheme might be useful as an extra service to log in when the access point for the sought service is unsecure.

Innehåll

Bakgrund

Verifiering på datorer är en viktig del av det samhälle vi lever i. Det beror på att man ofta har ett personligt konto knutit till en tjänst för att kunna spara inställningar, tillgänglighet av program, andra tjänster samt personlig data. Verifikation, att du faktiskt är den du utger dig för att vara, är till för att skydda tjänsten mot attacker och att fel personer ska får tag i dina data eller tillgänglighet till dina tjänster. Den här rapporten tar upp och går igenom ett speciellt sätt att verifiera sig kallat kognitivt verifikationsschema. I rapportens första del går vi igenom vanliga verifikationssätt med dess för och nackdelar samt olika sätt att knäcka dessa. I den senare delen går vi igenom de tankar som finns runt kognitivt verifikationsschema och en enkel implementering.

Olika verifikationssätt

Det finns många olika verifikationssätt på datorer och nedan tar rapporten upp några av dessa med fördelar och nackdelar. Att verifiera sig kallas ofta för att logga in.

PIN-kod

Exempel: 1234
Används i mobiltelefoner och till bankomater. Verifikationen består i att du anger siffror, i given ordning, oftast fyra stycken för att verifiera dig. Fördelen är att det är kort och lätt att lära sig fyra siffror. Nackdelen är att det bara finns 10^4=10000 kombinationer att välja mellan.

Textbaserade lösenord

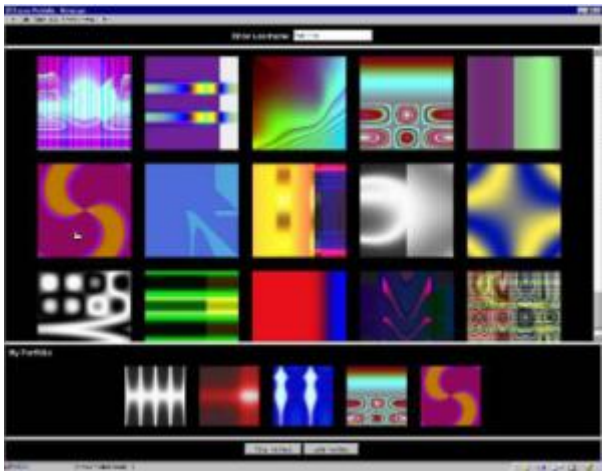
Exempel: aBc1
När man verifiera sig till en webbsida är nog den vanligaste formen av verifikation att använda sig av textbaserade lösenord. Textbaserade lösenord är helt enkelt ett ord, flera ord eller några tecken som du får memorera för att verifiera dig själv. Detta är även det mest använda verifieringssättet på arbetsplatser eller hemdatorer där du får skriva in ett textbaserat lösenord för att få tillgång till datorn. Fördelen är att man kan välja enkla lösenord som är lätta att komma ihåg. Längre lösenord är säkrare men svårare att komma ihåg. Nackdelen är att korta lösenord med få tecken har få kombinationer.

Bild med teckenserie

Figur 1 En teckenserie.

Används ofta när man ska skapa ett nytt konto hos en tjänst. Man får helt enkelt en bild som innehåller förvridna tecken som man får skriva ner, detta används för att en dator inte ska kunna lösa denna verifikation i sig själv. Fördelen är att det är svårt för en dator att avläsa teckenserien. Nackdelen är att det även kan vara svårt för en människa att avläsa tecknen

Grafiska verifikationer



Figur 2 En grafisk verifikation.

Grafiska verifikationer, alltså verifikationer som använder sig mer av bilder och andra visuella detaljer, blir allt vanligare på grund av den minskade säkerheten med vanliga textbaserade lösenord. Flera exempel diskuteras i [3]. I exemplet ovan ska man hitta de fem bilderna i nedre delen bland de många andra bilderna ovanför och markera dessa. Dina fem bilder visas under en inlärningsperiod sedan ska du av säkerhetsskäl komma ihåg dem själv. Fördelen är att det är lättare att komma ihåg bilder för användaren än ord eller siffror [3]. Det här är ett relativt säkert sätt att verifiera sig. Nackdelen är att det tar lång tid att hitta alla bilder.

Sätt att knäcka verifikationen

Att knäcka verifikationer är en viktig del i att göra en verifikation säker, om det är lätt att knäcka verifikationen så är det inte en säker verifikation.

Brute-force (Totalsökning)

Brute-force är att man testar alla möjliga kombinationer på ett verifikationssätt. Pinkoder knäcks enkelt med denna teknik eftersom det bara finns 10^4=10000 kombinationer att välja mellan och brute-force testar helt enkelt alla dessa tills den hittar den rätta sifferkombinationen. Kortare textbaserade lösenord kan också knäckas på detta vis.

Phishing (Nätfiske)

Phishing är att man med hjälp av e-post eller webbsidor frågar efter en verifiering. Dessa ser oftast ut som originalen och kan därför vara svåra att upptäcka, men har man skrivit in sin

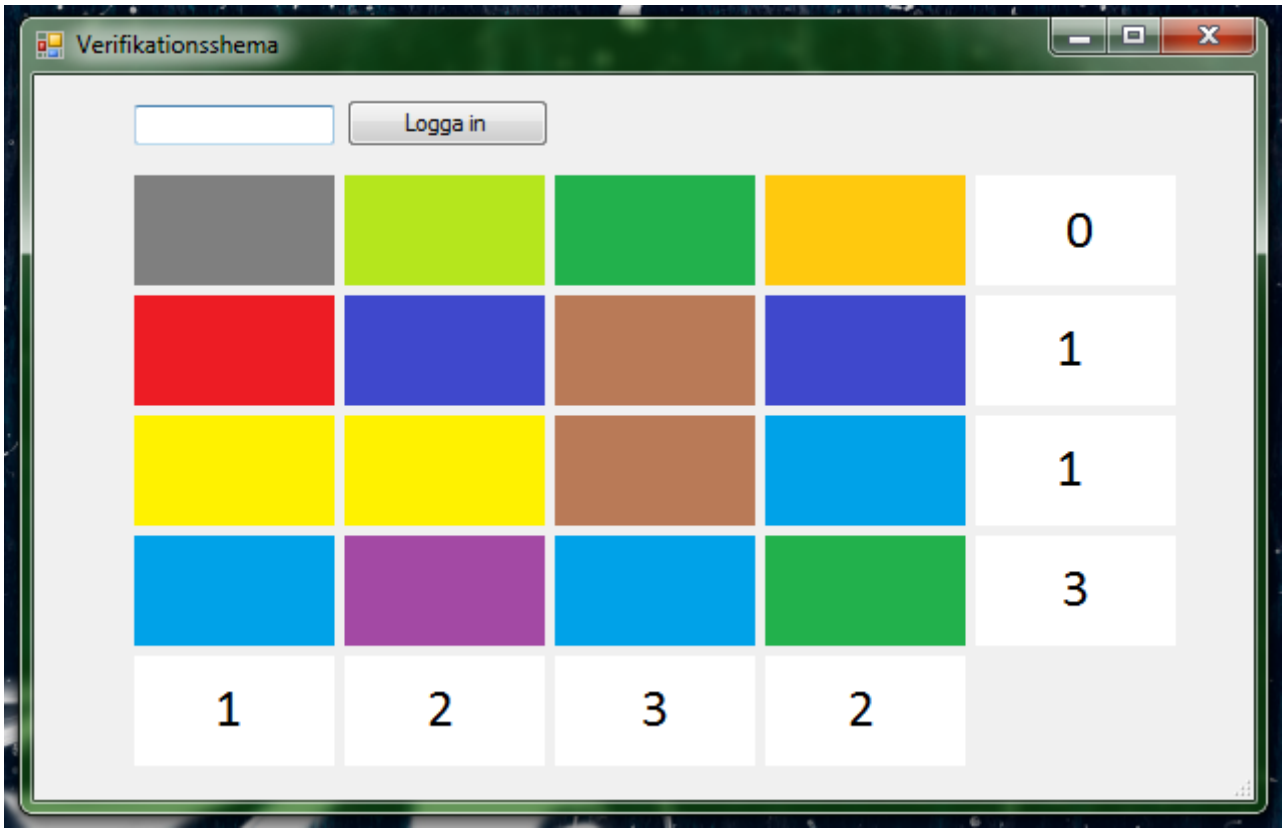
verifikation på dessa så är verifikationen knäckt. Pinkoder och textbaserade lösenord, oavsett längd, knäcks med denna metod.

Keyloggers (Tangentavläsare)

En keylogger är ett program som sparar alla knapptryckningar du gör på tangentbordet. Keyloggers kan lätt knäcka ett textbaserat lösenord eller Pinkod eftersom man har skrivit in det en gång och att det är samma lösenord som ska skrivas in nästa gång en verifikation ska ske. Mer avancerade keyloggers sparar även vart du har muspekaren och ibland även bilden du själv ser på skärmen, dessa kallas ofta för shoulder-surfing eftersom det är som om någon står bakom dig och ser allt du gör. Detta medför att även grafiska lösenord kan knäckas eftersom bildkoden inte ändras till nästa gång du ska verifiera dig.

Kognitivt verifikationsschema

En verifikation som verkar lovande är kognitivt verifikationsschema som, liksom exemplet ovan med grafisk verifikation, går ut på att du ska memorera ett antal bilder. Sättet du sedan verifierar dig på är annorlunda. Det här systemet är svårare att knäcka eftersom den som vill knäcka verifikationen måste analysera hela skärmen och till det krävs det mer datorkraft. Människor har lättare att komma ihåg bilder och detaljer än en större mängd tecken som i längre textbaserade lösenord [3].



Figur 3 Ett Kognitivt verifikationsschema.

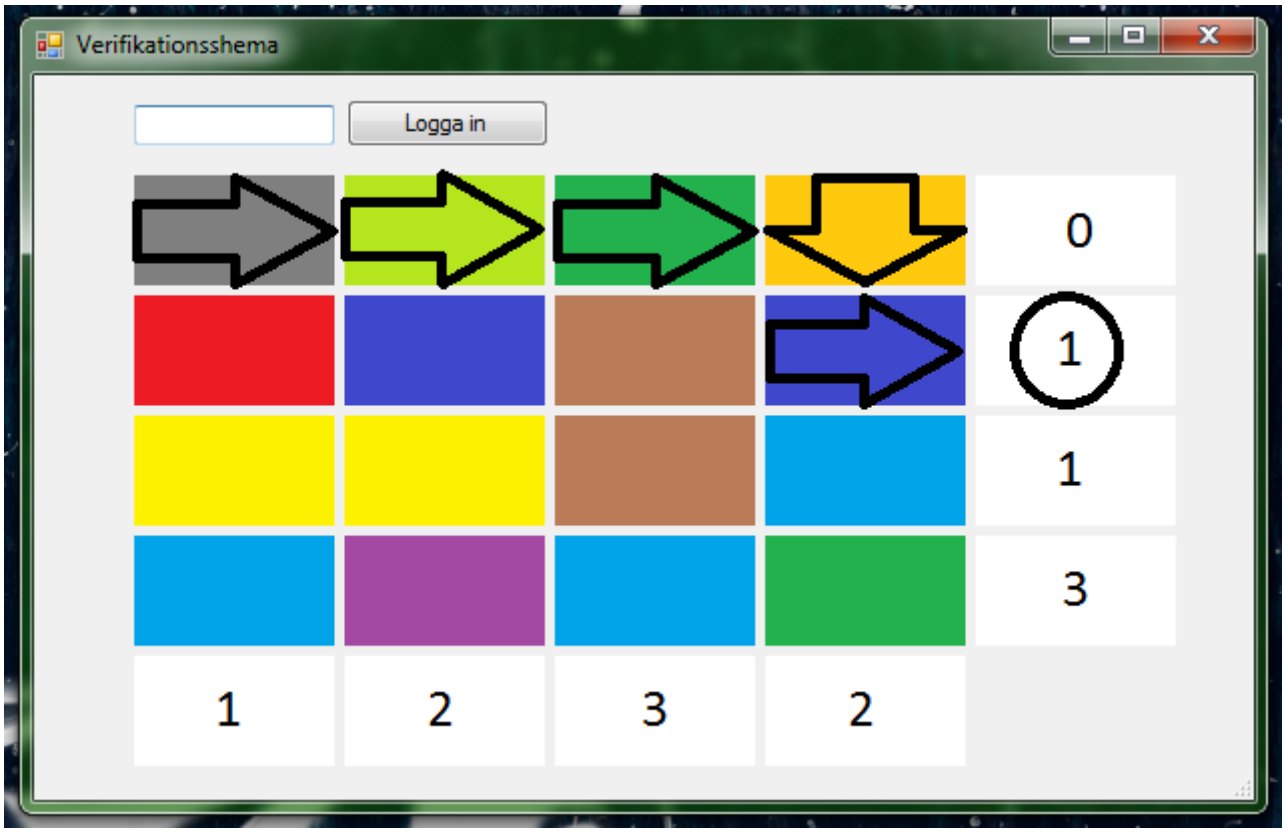
Du får upp en stor matris av bilder där några är dina och andra inte är det, du tittar sedan på bilden längst upp till vänster och ser om detta är din bild. Om det är det så går du en ruta ner i matrisen annars går du istället till höger. Sedan gör du på detta sätt även för nästa bild, om

den är din går du nedåt annars åt höger. Till slut har du kommit ut ur matrisen och där står det en siffra som blir svaret på utmaningen.

I detta enkla exempel är dessa 4 bilder de rätta.

Figur 4 Visar bilderna som är de rätta.

Med detta kan man se att det rätta svaret blir nummer 1.



Figur 5 Visar lösningen på vårt verifikationsschema exempel.

Detta är som sagt endast en enkel implementering som har gjorts för denna rapport, koden finns som bilaga 1. Man gör sedan om övningen ett antal gånger tills sannolikheten att man inte bara har gissat rätt nummer är tillräckligt sannolik. I detta exempel kan svaret vara mellan 0 och 3 och därför behövs 10 inloggningar för att höja verifikationssannolikheten. Om du inte följer din väg på skärmen med musen så görs allt jobb i ditt huvud och kan alltså inte spåras av en keylogger eller av shoulder-surfing. Det gör att detta sätt att verifiera sig är mer säkert en ovan nämnda exempel på verifieringar. Anledningen till att detta ännu inte används är att man med sannolikhetslära efter ett tag kan komma fram till vilka bilder som är dina, detta eftersom bilderna slumpas fram och hamnar på olika ställen och att man därför kan utesluta vissa bilder och inkludera andra. Ett annat problem är den långa inlärningstiden som krävs vid större utmaningar, som nämns i [2]. Så även denna verifikation är inte säker i nuläget även om det krävs flera inloggningar för att lista ut ditt lösenord så krävs det i nuläget för få

inloggningar för att man ska kunna kalla det ett säkert verifikationssätt. Man bevisar detta i [1].

Problemformulering

Det finns två distinkta problem för kognitiv verifikation som skall tas upp i denna rapport. Det som först skall tas upp är hur man implementerar detta verifikationsschema. För det andra ska man beakta hur användbart det är med kognitiv verifikation.

Implementering

Programmet skall enkelt kunna skapa en verifikationsutmaning som på ett användbart sätt skall vara säkert samt inte vara allt för resurskrävande. Då tanken med denna typ av utmanig var avsedd för att kunna göra en säker inloggning via internet från en osäker dator t.ex. på ett internetcafé.

Praktisk tillämpning

Som med alla system finns det begränsningar på vad som är användbart i praktiken. Då att skicka en utmaning kräver mer än att memorera ett vanligt lösenord. Även i [1] tas det upp om detta vilket skall analyseras.

Implementering

Programmet består utav tre huvudsakliga delar:

1. Klienten
2. Inloggnings protokoll
3. Bilddatabasen

Den tredje delen är något som redan finns och används och kommer därmed bara beröras i kort omfattning utav hur en tänkt databas kan se ut. För klienten blir det lite mer utav vad den får göra och vad som den inte bör hantera.
Det viktiga i programmet är hur man hanterar skapandet utav utmaningen och hanterar lösnigen av den. Även om programmet som tas upp i rapporten är utav enkel natur så följer den principerna för kognitiv verifikation som [2] beskriver.

Klienten

Klienten är mera fristående från själva verifikationen, annars skulle ett spywareprogram få reda på delar av den gemensamma hemligheten. Den skall följa det utstakade protokollet och hantera visningen av bilderna.

Inloggningsprotokoll

Som det beskrevs om verifikationsschemat i bakgrund så skall man ur ett bildarkiv skapa en randomiserad utmaning. I [2] så antar man att man använder hela bildarkivet i utmaningen men man kan även ha en större databas än det set av bilder som behövs för en utmaning. Det skall inte finnas dubbletter då de kan visa sig bli en säkerhetsrisk, då de faktiska unika bilder som finns i utmaningen är färre. Så för att undvika detta använder man en lista där varje bild som väljs plockas ur mängden.
Innan man ger användaren utmaningen så krävs det att man beräknar vilket svar som blir korrekt genom att datorn får gå samma stig. När klienten har gett ett svar på utmaningen så skapar protokollet en ny och kontrollerar svaret på den gamla. Detta returneras inte tillbaka till klienten.

Bilddatabasen

Bilddatabasen sköter hanteringen av bilderna då det blir enklare att be om adressen till bilderna i den. Den del som hanterar alla bilderna kan vara allmän och behöver inte skyddas, det som är känsligt är hanteringen av vilka bilder som tillhör en specifik användare precis som med lösenord.

Analys av protokollet

Efter att ha gjort en implementering av kognitivt verifikationsschema kan man nu ta upp dess styrkor och brister. Det som ska analyseras är själva implementeringen och själva verifikationsschemat.

Implementeringen

Programmet skall kunna hantera flera klienter och får därmed inte ta för stora resurser då en utmaning skapas. I denna implementering har målet varit att den skall vara snabb dock är inte random en metod som använder lite resurser. För att kunna använda detta program bör man lösa det slumpmässiga bildvalet på ett bättre sätt, då man annars kommer behöva ha en random maskin på varje anslutning. Styrkan i programmet är att det är lätt att skapa en autentisering som skall försvåra för spyware att kapa ens identitet till en tjänst. Ändå är det enkla operationer vilket ger den en stor möjlighet till förbättringar.

Protokollet

Protokollet skall vara enkelt nog för att en användare skall kunna göra en inloggning helt utan hjälp från datorn. Detta ger det en svaghet då schemat blir begränsat i storlek, se [1] där de visar att det går lätt att knäcka detta protokoll. Detta på grund av att inlärning av lösenordet inte gör det möjligt att välja tillräckligt stora mängder bilder som användaren kan memorera. I deras rapport knäcker de protokollet genom att omvandla inloggningen till ett satisfierbarhetsproblem. Styrkan i detta protokoll är att man försvårar för en keylogger att få tag i en inloggning. Det är bättre än att den loggar ett vanligt lösenord på första försöket.

Slutsatser

Att göra en implementering utav ett kognitivt verifikationsschema kan göras med ganska små resurser. Dock kräver det en bra kapacitet på servern för att generera alla utmaningar. Det gör att det kan vara kostsamt att utnyttja detta sätt för tjänster som har stor tillströmning utav användare. Dock kan man erbjuda detta som en enklare inloggning då man inte sitter på en säker anslutning t.ex. sitter på internetcafé när man är ute och reser. Man har visat på att protokollet inte är säkert i [1]. I deras exempel knäcker de en inloggning på 60 lyckade utmaningar på en 8x10 matris. Det som också talar emot detta protokoll var att det krävdes träning på en säker plats för att garantera att den delade hemligheten inte blir avläst. Även om protokollet inte fungerar som det var tänkt i [2] så kan detta möjligtvis användas istället för engångskoder, där man istället för siffror får några bilder, detta skulle göra att phishing inte skulle lyckas med att komma åt engångskoderna eftersom det behövs mer än en inloggning för att komma fram till rätt bilder. Så om man utvecklar denna metod så kan detta vara ett komplement till dagens inloggningsmetoder.

Källor

(Philippe Golle 2007) Har ett mycket utarbetat argument om de brister systemet har, även om kognitiv verifikation inte håller helt mot attacker så är det bättre än tidigare metoder. Källans trovärdighet är bra då alla bevis bygger på enkla matematiska satisfierbarheter.
(Weisenhall 2006) Har en mycket utförlig beskrivning hur problemen skall lösas, med ordentliga bevis. Det som bör beaktas är att slutsatserna är bygga på en naive test metod vilket dock nämns i början på rapporten. Det finns mycket bra referat i rapporten som stöd för alla påståenden förutom den egna tesen som nämndes.
(Xiaoyuan Suo 2005) Går igenom olika exempel på grafiska lösenord och deras säkerhetsaspekter.
Källan känns legitimt på grund av att författaren inte själv kommit på dessa verifikationssätt utan läst andras och sammanställt dem.

Litteraturförteckning

[1] Philippe Golle, David Wagner. ”Cryptanalysis of a Cognitive Authentication Scheme (extended abstract).” *IEEE Security & Privacy.* Stanford University: IEEE, 2007. 66-70.
[2] Weisenhall, Daphna. ”Cognitive authentication schemes safe against Spyware.” *IEEE Symposium on Security and Privacy.* IEEE, 2006. 295-300.
[3] Xiaoyuan Suo, Ying Zhu G., Scott. Owen. *Graphical Passwords: A Survey.* Georgia State University: ACSAC, http://www.acsac.org/2005/papers/89.pdf, 2005.

Billaga 1

Main klassen:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
using cogAuth;

namespace Dkand
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
       [STAThread]
        static void Main()
        {

            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

Programmet som skapar GUI't

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Dkand
{
    public partial class Form1 : Form
    {
        PictureBox[,] bilder;
        int sida;
        int total;
        int[] ver;
```

```
 public Form1()
{
    this.StartPosition = FormStartPosition.CenterScreen;
  InitializeComponent();
  button1.Text = "Logga in";



    //Bestäm hur långa sidorna ska vara
  sida = 4;

    //Totalt antal bilder, Eftersom bilderna börjar på noll får man ta sista nummret + 1
  total = 12;



    //Temporär lösning på vilka bilder som är dina bilder
    //
    //
    int[] ver = {0, 1, 3, 10};
    this.ver = ver;

    //Skapar en n-matris med sida+1 som n
  bilder = new PictureBox[sida+1,sida+1];

    //Skalar om fönstret så att det passar din inställda sida
    this.Height = 100 + (sida + 1) * 60;
    this.Width = 100 + (sida + 1) * 105;

    //Skapar själva pictureBox objekten där sedan bilderna läggs
    for (int i = 0; i <= sida; i++)
  {
        for (int j = 0; j <= sida; j++)
        {
         bilder[i, j] = new PictureBox();
         bilder[i, j].Location = new System.Drawing.Point((50+i*105), (50+j*60));
         bilder[i, j].Size = new System.Drawing.Size(100, 55);
            this.Controls.Add(bilder[i, j]);
     }
  }

    //Skapar en Random för att kunna generera slumptal
  System.Random RandNum = new System.Random();



    //Slumpar ut bilderna och tagar dem för att man ska kunna identifiera dem
    for (int i = 0; i < sida; i++)
  {
        for (int j = 0; j < sida; j++)
        {
            int MyRandomNumber = RandNum.Next(total);
```

```
                bilder[i, j].Image = Image.FromFile(Application.StartupPath + "\\Bilder\\" + (MyRandomNumber)
+ ".png");
                    bilder[i, j].Tag = (MyRandomNumber);
                }
            }


            //Lägger till siffrorna som sedan blir det tal som ska skrivas som svar
            for (int i = 0; i < sida; i++)
            {
                int MyRandomNumber = RandNum.Next(4);

                bilder[i, sida].Image = Image.FromFile(Application.StartupPath + "\\Bilder\\tal" +
(MyRandomNumber) + ".png");
                bilder[i, sida].Tag = ("tal" + (MyRandomNumber));

                MyRandomNumber = RandNum.Next(4);

                bilder[sida, i].Image = Image.FromFile(Application.StartupPath + "\\Bilder\\tal" +
(MyRandomNumber) + ".png");
                bilder[sida, i].Tag = ("tal" + (MyRandomNumber));
            }


        }

        private void button1_Click(object sender, EventArgs e)
        {
            //Kollar om verifieringen blir samma som svaret
            if (Verifiera() == ("tal" + textBox1.Text))
            {
                //Om det var korrekt, stäng ner programet
                MessageBox.Show("Det var det rätta nummret", "Mycket riktigt", MessageBoxButtons.OK,
MessageBoxIcon.Information);
                this.Close();
            }
            else
            {
                //Om det var fel, gör ett nytt verifikationsschema
                MessageBox.Show("Pröva igen", "Tyvärr fel", MessageBoxButtons.OK, MessageBoxIcon.Error);
                Form1 igen = new Form1();
                igen.StartPosition = FormStartPosition.CenterScreen;
                this.Hide();
                igen.ShowDialog();
                this.Close();
            }
        }

        private String Verifiera()
        {
            int i = 0;
            int j = 0;
            int kolla;
```

```
            while(i < sida && j < sida)
        {
            //Kontrollera tagen på bilden
           kolla = Convert.ToInt32(bilder[i,j].Tag.ToString());
            //Om bilden tillhör lösenordet gå nedåt
            if (ver.Contains(kolla))
           {
             j++;
            }
            //Annars gå åt höger
            else { i++; }
        }
          //Returnera namnet på den sifferbild som verifikationen stannade på
          return bilder[i, j].Tag.ToString();
      }
   }
}
```