Brian Kiser
November 2010

# Vigilant

C# 2.5

Commonwealth of Kentucky
Frankfort, Kentucky

Table of Contents

## 1.0 Work Sample Description

Vigilant was written as a warning system to quickly notify office personnel in the event of an emergency. It was requested by Labor Cabinet management after the Cabinet for Health and Family Services (CHFS) demonstrated an application called “Red/Blue Alert” that they used to notify users in case of an unauthorized intruder in the building.

The CHFS version had several flaws, however, such as requiring users to be set up individually and the need for a component to be installed on the server. I wanted network computers to communicate directly, without a database or a complicated software install. Vigilant provides all the same warning functionality and more, yet without the disadvantages of “Red/Blue Alert.” Users do not need to be set up and the only installed necessary is a program on the desktop PC. That install can even be performed by going to a link, so installation is minimal. Vigilant quietly resides in the system tray when running.

Vigilant is currently being used by CHFS for their Department for Community Based Services (DCBS), and has been distributed throughout Kentucky’s 120 counties.

## 2.0 Skills Demonstrated

### 2.1 Software development competency using C# and sockets

Vigilant is a small desktop application that uses UDP broadcasting technology in order to send warning alerts to all office PCs. The broadcast technology used is exactly the same as is used in chat rooms across the Internet. In fact, one of the test phrases for Vigilant involved creating a chat room client and sending messages back and forth. A small part of the program also implemented threading.

TCP broadcasting was also considered initially, due to the fact that UDP is not considered “reliable.” Messages can be lost using UDP. However, the advantages outweighed the disadvantages. TCP required the program to know the IP of the receiving computer. UDP does not maintain connection state, and thus is more efficient. UDP also does not introduce any delay to establish a connection, thus messages are sent instantly.

This application was written in C# with no database connectivity.

### 2.2 Ability to work with multiple managers as stakeholders

Vigilant was developed at the request of upper-management, but with user input from both lower- and upper-management as stakeholders. At times, this presented a unique set of difficulties.

### 2.3 Ability to write complex code under pressure

Vigilant uses UDP socket technology and was written in a relatively short amount of time. Research into sockets was performed in order to satisfy under requirements, but also meet reliability standards. Vigilant was requested by upper-management, and an appropriate sense of urgency was required.

## 3.0 Difficulties Encountered

Management instruction differed and sometimes conflicted, and this had to be resolved by getting everyone on the same page. At times, both levels of management felt strongly about specific issues. Had this been a larger project, it would have been even more important to resolve these type of issues quickly.

Understanding and coding sockets and UDP broadcasting was a substantial effort, but manageable due to the small scope of the project.

The project was highly successful. Vigilant is currently in use on all ~250 desktops at the Kentucky Labor Cabinet and is part of the required “active shooter” training program, as well as in use in all 120 counties in Kentucky for CHFS’s offices.

## 4.0 Screen Shots

Please see Appendix A for enlarged versions.

### 4.1 The main screen

The main screen in Vigilant only shows when the user clicks the system tray icon. Initially the idea was for an intruder alert program only, but upper-management decided that there should be alerts added for two other emergencies, like fire and weather. The screen is pleasant to the eye with gradient colors and the buttons are

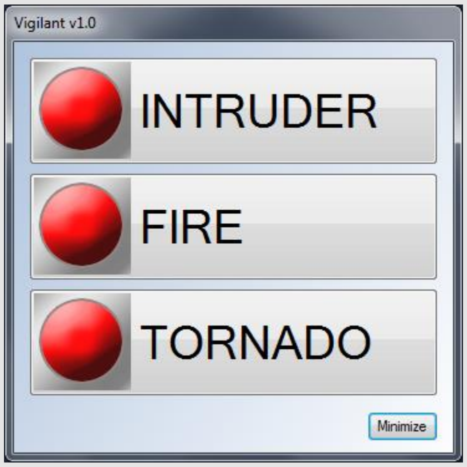
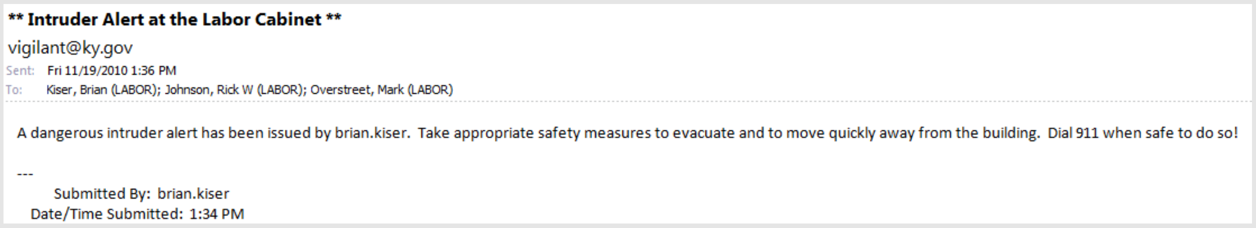


**Figure B – Vigilant's main screen**

prominent.

### 4.2 Email Notification

In addition to a screen alert, Vigilant can notify users via email. Management felt that this would be useful for users who are currently out of the office. At one point, it was considered a "quiet" method to notify employees, but management decided they wanted a visual and audio alert added as well. Users may not check their email often, so using email alone, there was the fear that users would not receive the alert.



**** Intruder Alert at the Labor Cabinet ****
vigilant@ky.gov
Sent: Fri 11/19/2010 1:36 PM
To: Kiser, Brian (LABOR); Johnson, Rick W (LABOR); Overstreet, Mark (LABOR)

A dangerous intruder alert has been issued by brian.kiser. Take appropriate safety measures to evacuate and to move quickly away from the building. Dial 911 when safe to do so!

---
Submitted By: brian.kiser
Date/Time Submitted: 1:34 PM

**Figure A – A copy of the Email Notification**

### 4.3 Icon in the System Tray

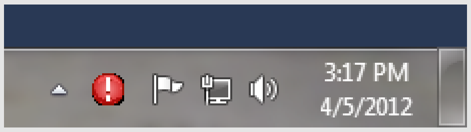


**Figure B – Vigilant in the system tray**

Vigilant stays in the system tray out of the way, but is easy to access should the need arise.

## 5.0 Code Listing

There were multiple versions of Vigilant. The CHFS version is slimmed down, contains a “stealth” warning, and does not include verbal messages or extra buttons for weather or fire. It is used exclusively in the case of an unauthorized intruder on the premises. The version below is the more elaborate version written for Labor that includes verbal warnings and extra emergency buttons. Due to the code being written under deadline by upper-management, documentation is sparse.

### 5.1 frmVigilant.cs

The main block of code sets up a listener on the specified port, defines the user interface, creates the context menu in the tray (which includes a help screen for new users), and various other things that are necessary. Most importantly, it broadcasts the appropriate UDP warning signal.

```
/*
 * Vigilant
 * Brian Kiser
 * 10/2010
 */

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;
using System.Diagnostics;                   // for Process.Start
using System.Net.Sockets;
using SocketLibrary;
using System.IO;
using System.Reflection;


namespace Vigilant {

      // This delegate enables asynchronous calls for setting the text property on a TextBox control.
      delegate void SetTextCallback(string text, string hostname, string IP, string MachineName, string TimeStamp);

      public partial class frmVigilant : BaseFormGradient {

            // Define the menu.
            private ContextMenu MyMenu= new ContextMenu();
            private MenuItem about    = new MenuItem ("About Vigilant");
            private MenuItem divider  = new MenuItem ("-");
            private MenuItem help     = new MenuItem ("Help");
            private MenuItem exitApp  = new MenuItem ("Exit");

            private int SocketNumber = 54256;      // This is the socket Vigilant will be using for communication.

            MySocket sock = new MySocket();

            public frmVigilant() {
                  InitializeComponent();
                  sock.OnMessageReceived += new MySocket.MessageReceivedHandler (sock_OnMessageReceived);

                  // Form has to be minimized here.  If I try it in the form load,
                  // it minimizes above the system tray and looks funky.
                  WindowState = FormWindowState.Minimized;
            }


            void sock_OnMessageReceived(object sender, SocketLibraryEventArgs e) {
                  SetText (e.Message, e.HostName, e.SenderIP, e.MachineName, e.TimeStamp);
```

```
}


// -------------
// -  ExitApp  -
// -------------
private void ExitApp (object sender, System.EventArgs e) {

      appIcon.Dispose();
      Application.Exit();
}


// This method demonstrates a pattern for making thread-safe calls on a Windows Forms control.
// If the calling thread is different from the thread that created the TextBox control, this method
// creates a SetTextCallback and calls itself asynchronously using the Invoke method.
// If the calling thread is the same as the thread that created the TextBox control, the
// Text property is set directly.

private void SetText (string text, string hostname, string IP, string machinename, string timestamp) {

      if (!text.Equals ("")) {

            if (this.InvokeRequired) {
                SetTextCallback d = new SetTextCallback (SetText);
                this.Invoke (d, new object[] { text, hostname, IP, machinename, timestamp });
            } else {
                  Message m = new Message (text, hostname, IP, machinename, timestamp);

                  // Don't display the warning to the person broadcasting it.
                  if (!sock.GetIP ().Equals (IP)) {
                        frmAlert alert = new frmAlert (m);
                        alert.Show();
                  }
            }

      }
}


private void btnCancel_Click (object sender, EventArgs e) {
      this.WindowState = FormWindowState.Minimized;
}


// ---------------
// -  Form Load  -
// ---------------

private void frmVigilant_Load (object sender, EventArgs e) {

      ShowInTaskbar = false;                                      // Remove from taskbar.
      CreateMenu();

      try {
            sock.Listen (SocketNumber);
      } catch (Exception ex) {
            MessageBox.Show (ex.ToString());
      }
}


// ----------------
// -  CreateMenu  -
// ----------------
private void CreateMenu() {

      appIcon.Text = "Vigilant v1.0";

      // Place the menu items in the menu.
      MyMenu.MenuItems.Add (about);
```

```
        MyMenu.MenuItems.Add (divider);
        MyMenu.MenuItems.Add (help);
        MyMenu.MenuItems.Add (exitApp);
        appIcon.ContextMenu = MyMenu;

        // Show the system tray icon.
        appIcon.Visible = true;

        // Attach event handlers.
        exitApp.Click += new EventHandler (ExitApp);
        help.Click    += new EventHandler (ShowHelp);
        about.Click   += new EventHandler (AboutScreen);
}


// -----------------
// -  AboutScreen  -
// -----------------
private void AboutScreen (object sender, System.EventArgs e) {

        frmAbout about = new frmAbout();
        about.TopMost = true;                           // keep it on top
        about.ShowDialog();                             // make it modal
}


// --------------
// -  ShowHelp  -
// --------------
private void ShowHelp (object sender, System.EventArgs e) {

        System.Reflection.Assembly asm = System.Reflection.Assembly.GetExecutingAssembly();
        System.IO.Stream stream = asm.GetManifestResourceStream ("Vigilant.Documentation.readme.doc");
        byte[] m_Bytes = new byte [2048];               //2048 considered as int
        System.IO.FileStream sb = new System.IO.FileStream (Directory.GetCurrentDirectory() +
                "\\readme.doc", System.IO.FileMode.Create);

        int j = stream.Read (m_Bytes, 0, m_Bytes.Length);

        while (j > 0) {
                sb.Write (m_Bytes, 0, j);
                j = stream.Read (m_Bytes, 0, m_Bytes.Length);
        }       //copy it with original bytes

        sb.Close();
        System.Diagnostics.Process.Start ("readme.doc");
}


// -----------------
// -  Form Resize  -
// -----------------

private void frmVigilant_Resize (object sender, EventArgs e) {

        // Remove the taskbar button if the program is minimized.  It should only appear in the tray.
        if (FormWindowState.Minimized == WindowState) {
                appIcon.Visible = true;
                //notifyIcon1.ShowBalloonTip (500);
                Hide();

        } else if (FormWindowState.Normal == this.WindowState) {
                appIcon.Visible = false;
        }
}


private void appIcon_DoubleClick(object sender, EventArgs e) {

        // Restore the application when the icon is double-clicked.
        Show();
        WindowState = FormWindowState.Normal;
```

```
            }

            private void frmVigilant_FormClosed(object sender, FormClosedEventArgs e) {

                  // May not be necessary.
                  appIcon.Dispose();
                  Application.Exit();
            }


            // -----------------------
            // -  big button clicks  -
            // -----------------------
            private void btnIntruder_Click (object sender, EventArgs e) {

                  this.WindowState = FormWindowState.Minimized;
                  MySocket sock = new MySocket();
                  sock.SendMessage ("INTRUDER", SocketNumber);
            }
            private void btnFire_Click(object sender, EventArgs e) {
                  this.WindowState = FormWindowState.Minimized;
                  MySocket sock = new MySocket();
                  sock.SendMessage ("FIRE", SocketNumber);
            }
            private void btnTornado_Click(object sender, EventArgs e) {
                  this.WindowState = FormWindowState.Minimized;
                  MySocket sock = new MySocket();
                  sock.SendMessage ("TORNADO", SocketNumber);
            }

      }           // class
}                 // namespace Vigilant
```

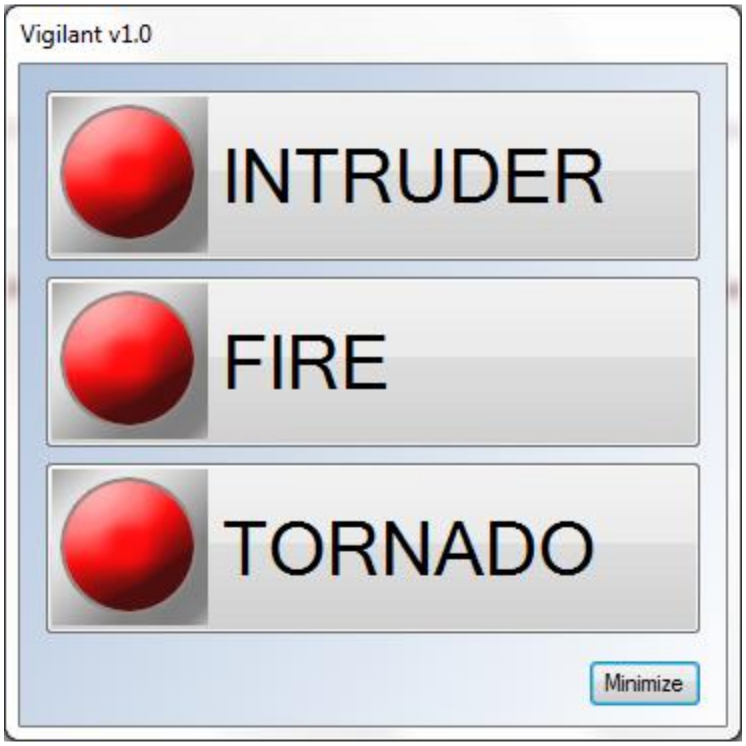
## 5.2 frmAlert.cs

This form receives the type of alert from frmVigilant.frm and displays the appropriate warning.  It's simple code.

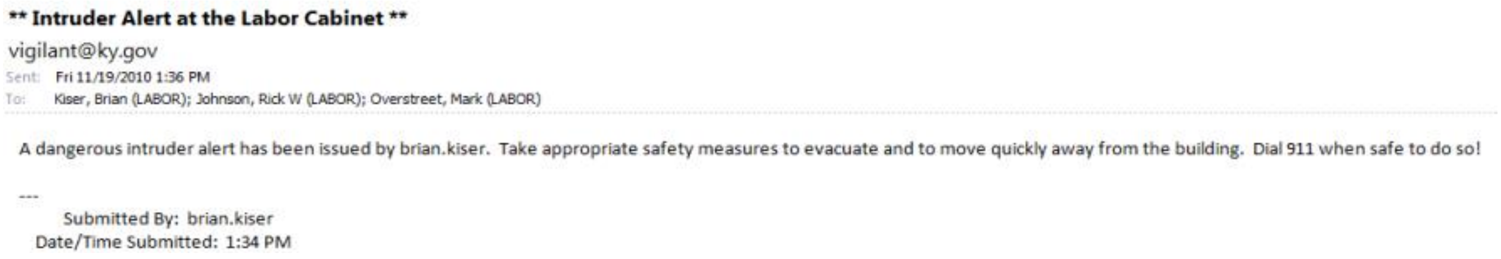
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Vigilant {

    public partial class frmAlert : BaseFormGradient {

          private Message _Message = new Message();

          private enum AlertType {
                Intruder,
                Fire,
                Tornado
          }
          private AlertType at;

          // This override is necessary so that the form can be passed parameters.
          public frmAlert (Message msg) : this () {
                _Message = msg;

                switch (_Message.MessageText) {
                      case "INTRUDER" :
                        lblTitle.Text = "INTRUDER ALERT";
                        _Message.MessageText = "A dangerous intruder alert has been issued by " +
                              _Message.HostName + ".  Take appropriate safety measures to evacuate
                              and to move quickly away from the building. Dial 911 when safe to do
                              so!";
                        at = AlertType.Intruder;
                        break;
                  case "FIRE":
                        lblTitle.Text = "FIRE";
                        _Message.MessageText = "Warning:  A fire alarm has sounded in the 127
                        South Building.  Take appropriate measures to evacuate the building
                        immediately! This is not a drill!\n\nSent from the desk of " +
                        _Message.HostName + ".";
                        at = AlertType.Fire;
                        break;
                  case "TORNADO":
                        lblTitle.Text = "TORNADO";
                        _Message.MessageText = "The national weather service has issued a TORNADO
                        WARNING for Franklin County.  Take Shelter immediately!  This is not a
                        drill.\n\nSent from the desk of " + _Message.HostName + ".";
                        at = AlertType.Tornado;
                        break;
            }

            lblMessage.Text = _Message.MessageText;

      }

      public frmAlert () {
          InitializeComponent();
      }

      private void btnClose_Click(object sender, EventArgs e) {
            this.Close();
      }

      // ---------------
      // -  Activated  -
      // ---------------
      // This event fires after the form is loaded.  I wanted to make sure the text is up on the
      // form before I played the sound.
      private void frmAlert_Activated (object sender, EventArgs e) {

            //Console.WriteLine ("ACTIVATED EVENT...");

            switch (at) {
                  case AlertType.Intruder:
                        Winmm.PlayWavResource ("weatherwarning.wav");
                        System.Threading.Thread.Sleep (3000);
                        Winmm.PlayWavResource ("Intruder Alert.wav");
                        //Console.WriteLine ("played intruder sound");
                        break;
                  case AlertType.Fire:
                        Winmm.PlayWavResource ("weatherwarning.wav");
                        System.Threading.Thread.Sleep (3000);
                        Winmm.PlayWavResource ("Fire Warning.wav");
                        //Console.WriteLine ("played fire sound");
                        break;
                  case AlertType.Tornado:
                        Winmm.PlayWavResource ("weatherwarning.wav");
                        System.Threading.Thread.Sleep (3000);
                        Winmm.PlayWavResource ("Tornado Warning.wav");
                        //Console.WriteLine ("played tornado sound");
                        break;
                        }
                }
        }
}
```

**6.0 Appendix A**

6.1 Alert Screen



6.2 Email Notification



**** Intruder Alert at the Labor Cabinet ****

vigilant@ky.gov

Sent: Fri 11/19/2010 1:36 PM

To: Kiser, Brian (LABOR); Johnson, Rick W (LABOR); Overstreet, Mark (LABOR)

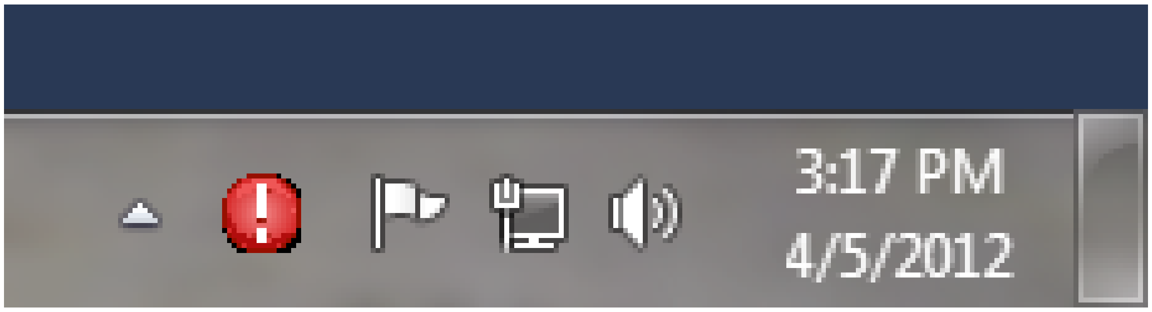
A dangerous intruder alert has been issued by brian.kiser. Take appropriate safety measures to evacuate and to move quickly away from the building. Dial 911 when safe to do so!

---

Submitted By: brian.kiser

Date/Time Submitted: 1:34 PM

6.3 System Tray


3:17 PM
4/5/2012