VOLUME: 3
ISSUE: 1
JANUARY - APRIL, 2019

PRIZREN SOCIAL SCIENCE JOURNAL

ISSN: 2616-387X
DOI: 10.32936/PSSJ.V3I1

CREATING PROTOTYPE VIRUS - DESTROYING FILES AND TEXTS ON ANY COMPUTER

Prof. Ass. Naim BAFTIU*, Prof. Asoc. Samedin KRRABAJ**

**Prizren University "UKSHIN HOTI, Faculty of Computer Science, naim.baftiu@uni-prizren.com*

***Prizren University "UKSHIN HOTI, Faculty of Computer Science, samedin.krrabaj@uni-prizren.com*

Article history:

Accepted 21 April 2019
Available online 30 April 2019

Keywords:
Component,
Virus,
File,
C# Programming,
Visual Studio.

A b s t r a c t

Problem description -When we study how viruses work and prevent them, we've developed a very simple application where we can see a prototype of a virus and virus function, as well as neutralizing a file if we want to break it down its structure at the level of the bits Purpose-Understand how a virus works by programming it in a high programming language - In our case, the C # programming language with the Visual Studio program that uses the .Net Framework. With the Windows Form Application module, the same application we are creating can also use it to neutralize a sentence if we know it is infected by interfering with the file we set up itself and by disrupting the system his Binary.

Introduction

Since computer science is a wider and more sub-direction, viruses and defense play a very important role in science's development. Every day we are confronted with viruses, and their prevention is essential, because every day more and more virulent, so destructive, viruses develop.

1. Research Methodology

The research methods will be based on prior virus research and their understanding by renowned international authors in this regard and a lot of transcript input, and also based on the literature on the basics of programming where we will write a key method for our application.

2. Content of the Work

Viruses as we know are all other applications, but the purpose of which is badly intended. From the early virus and programming knowledge, the following application is programmed very simple and very short but relatively effective and a very good picture of how viruses work.

We know that C # programming language is object-oriented, so programming will also be based on objects, by visualizing them with the Windows Form Application module.

C# (pronounced "See Sharp") is a simple, modern, object-oriented, and type-safe programming language. C# has its roots in the C family of languages and will be immediately familiar to C, C++, and Java programmers. C# is standardized by ECMA International as the ECMA-334 standard and by ISO/IEC as the ISO/IEC 23270 standard. Microsoft's C# compiler for the .NET Framework is a conforming implementation of both of these standards.

C# is an object-oriented language, but C# further includes support for component-oriented programming. Contemporary software design increasingly relies on software components in the form of self-contained and self-describing packages of functionality. Key to such components is that they present a programming model with properties, methods, and events; they have attributes that provide declarative information about the component; and they incorporate their own documentation. C# provides language constructs to directly support these concepts, making C# a very natural language in which to create and use software components.

Several C# features aid in the construction of robust and durable applications: Garbage collection automatically reclaims memory occupied by unused objects; exception handling provides a structured and extensible approach to error detection and recovery; and the type-safe design of the language makes it impossible to read from uninitialized variables, to index arrays beyond their bounds, or to perform unchecked type casts.

C# has a unified type system. All C# types, including primitive types such as int and double, inherit from a single root object type. Thus, all types share a set of common operations, and values of any type can be stored, transported, and operated upon in a consistent manner. Furthermore, C# supports both user-defined reference types and value types, allowing dynamic allocation of objects as well as in-line storage of lightweight structures.

To ensure that C# programs and libraries can evolve over time in a compatible manner, much emphasis has been placed on versioning in C#'s design. Many programming languages pay little attention to this issue, and, as a result, programs written in those languages break more often than necessary when newer versions of dependent libraries are introduced. Aspects of C#'s design that were directly influenced by versioning considerations include the separate virtual and override modifiers, the rules for method overload resolution, and support for explicit interface member declarations.

The rest of this chapter describes the essential features of the C# language. Although later chapters describe rules and exceptions in a detail-oriented and sometimes mathematical manner, this chapter strives for clarity and brevity at the expense of completeness. The intent is to provide the reader with an introduction to the language that will facilitate the writing of early programs and the reading of later chapters.

In our case we have:

♣First, we will create the function for selecting the desired file so that it can be processed, so the best method is to select a Button and a Text Box. (Fig.1)

Figure 1. Text Box

Fig.1. Button **"**Assign File**"** in program we called "**btnAssign**" ethier Textbox **"txtAssign"**

♣In the background will be inserted the **openFileDialog** module that will be used as openFileDialog1 which will help you as the file selection window. (Fig2).

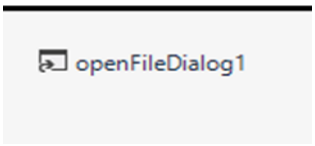

Figure 2. File Dialog

We will also add a "checkbox" which we will call "checkbox1" in the program and in the **"Location"** interface only to verify the location selection, whether it is automatic through the module that we will create or the manual written text.

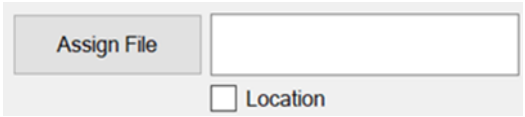

Figure 3. Text Box Location

♣ In the program we will have a major variable that we will manipulate and we will call it the **"fileName"** of string type **(text).**

The following code will be the file selection mechanism where the **"event"** code will be modified if you press the button:

3. Program structure

The key organizational concepts in C# are ***programs***, ***namespaces***, ***types***, ***members***, and ***assemblies***. C# programs consist of one or more source files.

Programs declare types, which contain members and can be organized into namespaces. Classes and interfaces are examples of types. Fields, methods, properties, and events are examples of members. When C# programs are compiled, they are physically packaged into assemblies. Assemblies typically have the file extension .exe or .dll, depending on whether they implement ***applications*** or ***libraries***.

4. Types and Variables

There are two kinds of types in C#: ***value types*** and ***reference types***. Variables of value types directly contain their data whereas variables of reference types store references to their data, the latter being known as objects. With reference types, it is possible for two variables to reference the same object and thus possible

for operations on one variable to affect the object referenced by the other variable. With value types, the variables each have their own copy of the data, and it is not possible for operations on one to affect the other (except in the case of ref and out parameter variables).

C#'s value types are further divided into ***simple types***, ***enum types***, ***struct types***, and ***nullable types***, and C#'s reference types are further divided into ***class types***, ***interface types***, ***array types***, and ***delegate types***.

in our case we have:

```
private void btnAssign_Click(object sender, EventArgs e)
    {
      DialogResult result = openFileDialog1.ShowDialog();
      if (result == DialogResult.OK)
      {
        fileName = openFileDialog1.FileName;
        txtAssign.Text = fileName;
      }

    }
```

What happens if we click the button, then we will develop the selection mechanism where the window will be displayed as a result, and we have a conditional if we finish the selection, variably earlier in the filenames declaration, will get the value of the window for selection, at the same time **textBoxi txtAssign** will get the value of the file name.

At the same time, we also develop the code for the location confirmation **checkbox**

```
private void checkBox1_CheckedChanged(object sender, EventArgs e)
    {
      if (checkBox1.Checked) txtAssign.ReadOnly = true;
      else checkBox1.Checked = txtAssign.ReadOnly = false;
    }
```

From the following code we can notice that we have the conditionality if we have done the Tick checkbox, the textbox attribute will go to ReadOnly, respectively unchanged, and if we did not tick, the textbox would stay the same as before unchanging.

We have now developed the selection mechanism, and the next file destruction process will occur in the main variable of FileName.

```
void destroyFile(string fileName)

{
        byte[] bytes = System.IO.File.ReadAllBytes(fileName);
        byte[] oldBytes = bytes;
        Random rnd = new Random();
        int num = rnd.Next(0, 255);

        for (int i = 0; i < bytes.Length; i++)
        {
          if (bytes[i] % 2 == 0)
          {
            bytes[i] = (byte)rnd.Next(0, (int)(bytes[i] * 1.618 < 256.0 ?  bytes[i] * 1.618 : 255));
          }
        }

        File.WriteAllBytes(fileName, bytes);
    }
```

We will have a method which as input parameter will have the fileName variable

```
void destroyFile(string fileName)
```

Then we will create a variable byte bytes where all of the file bytes will be scanned by the **Syste.IO.File.ReadAllBytes () method;**

```
byte[] bytes = System.IO.File.ReadAllBytes(fileName);
```

Then a generic number will be generated, in the variable called **"rnd".**

```
int num = rnd.Next(0, 255);
```

We will have a loop **(rewrite**) where we will use it to insert Random numbers into the desired file for decay. The length of the loop will be up to the end of the file. The data will be entered through the kite ordering conditionality **(proportional to two)** and will be entered through the loop control.

```
(int i = 0; i < bytes.Length; i++)
                {
                    if (bytes[i]
                % 2 == 0)
                {
            bytes[i]    =    (byte)rnd.Next(0,
        (int)(bytes[i] * 1.618 < 256.0 ? bytes[i] *
        1.618 : 255));
        }
}
```

Once the process is completed, **"FileWriteAllBytes"** will write to the file those generated numbers **(Bytes)**, where as input parameters will have the **fileName,** and generated bytes.

```
File.WriteAllBytes(fileName, bytes);
```

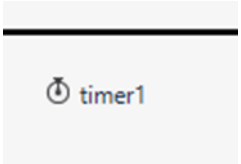

Figure 4. Timer

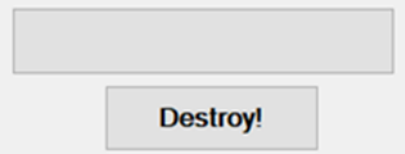

Figure 5. Progress Bar

Then we will create a "Timer" (Fig4) that we will use to display the **"ProgressBar"** and the main button **(btnDestroy)** to activate the previous method created for the demolition (Fig5).

Timer code:

```
private void timer1_Tick(object sender, EventArgs e)
    {
      progressBar1.Increment(10);
      if (progressBar1.Value == progressBar1.Maximum)
      {
        timer1.Stop();
        MessageBox.Show("The file is damaged
successfully!");
      }
    }
```

From the previous code we see how the progressbar changes during each **Timer beep,** where the progress bar will increase to the value of 10, and by conditioning we check that if the progressbar is filled, then the Timer will stop and display a window that will confirm the process.

The main button for breaking code:

```
private void btnDestroy_Click(object sender, EventArgs e)
    {
      timer1.Start();
      destroyFile(fileName);
    }
```

Where will start the previous Timer process, and will be called **"destroyFile"** where as input parameter will have the **fileName** variable that we have previously assigned.

Completed program:

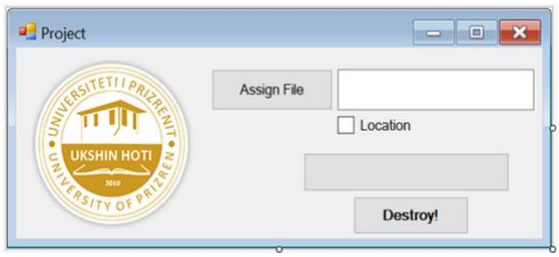

Figure 6. The completed code is found in the other document

Findings and contribution of the work

In this paper my main hypothesis is the idea of how easy it is to create a computer virus, but to destroy it. They can be used for Educational purposes but also for other purposes whatever they are in the digital world.

Through this project I will try to demonstrate how computer viruses are working through programming that will impact on expanding awareness of computer viruses awareness and prevention, which can lead to enormous destruction of very important data, be they personal, up to government levels.

References

1. Einführung in das Programmieren mit C# 4.0 2011 (Rev. 121022) ACHTUNG: Es ist ein ZIMK-

Manuskript zu C# 6.0 verfügbar: https://www.uni-trier.de/index.php?id=22777

2. Serazzi, Giuseppe; Zanero, Stefano (2004). "Computer Virus Propagation Models". In Calzarossa, Maria Carla; Gelenbe, Erol. *Performance Tools and Applications to Networked Systems*(PDF). Lecture Notes in Computer Science. Vol. 2965. pp. 26–50. Archived (PDF) from the original on 2013-08-18.

3. Pro .NET 4 Parallel Programming in C# January 2010, DOI: 10.1007/978-1-4302-2968-1 Adam Freeman

4. Jason Andress, Steve Winterfeld, Cyber Warfare USA, 2014 978-0-12-416672-1.

5. Lance Hayden, Ph.D. IT security Metrice, USA, 2010, ISBN: 978-0-07-171340-5.

6. Alfred J. Menezes, Paul C. Van Oorschot, Scott A. Vanstone, Handbook of Applied Gryptography. USA, 1997. ISBN: 978-0-8493-8523-0

7. Stallings, William (2012). *Computer security : principles and practice*. Boston: Pearson. p. 182. ISBN 978-0-13-277506-9.

8. Aycock, John (2006). *Computer Viruses and Malware*. Springer. p. 14. ISBN 978-0-387-30236-2.

9. Cohen, Fred (1984), *Computer Viruses – Theory and Experiments*, archived from the original on 2007-02-18

10. Bell, David J.; et al., eds. (2004). "Virus". *Cyberculture: The Key Concepts*. Routledge. p. 154. ISBN 9780203647059. Archived from the original on 2017-03-16.

11. Mark Ciampa, Security+ Guide to Network Security Fundamentals
12. USA, 2012, ISBN: 978-1-111-64017-0.