

# Forum:[Hack Forums](#)

Thread: [FileShreader secure file removal](#)

Immortal - July 24, 2015, 12:48 pm

First let it be know FileShreader is pronounced file she - reader. It was a joke. to be funny ha ha.

FileShreader.cs

PHP Code

```
<?php
```

```
/**
```

```
* @Author: Immortal
```

```
*/
```

```
using System;
```

```
using System.IO;
```

```
using System.Windows.Forms;
```

```
class FileShreader
```

```
{
```

```
    [STAThread]
```

```
    private static void Main(string[] args)
```

```
    {
```

```
        OpenFileDialog ofd = new OpenFileDialog();
```

```
        ofd.Filter = "All files *.*|*.*";
```

```
        if (ofd.ShowDialog() == DialogResult.OK)
```

```
        {
```

```
            string path = ofd.FileName;
```

```
            int offset = 0x0;
```

```
            byte[] tempfile = File.ReadAllBytes(path);
```

```
            byte[] data = new byte[tempfile.Length];
```

```
            Random r = new Random();
```

```
            byte[] key = new byte[256];
```

```
            r.NextBytes(key);
```

```
            tempfile = DexCryptMin.Encrypt(tempfile, key);
```

```
            if (
```

```
                System.Windows.Forms.MessageBox.Show(
```

```
                    "Are you sure you want to do this you will never get this file back again!", "Are you sure?",
```

```
                    MessageBoxButtons.YesNo) == DialogResult.Yes)
```

```
            {
```

```
                for (int i = 0; i < data.Length; i++)
```

```
                {
```

```
                    data[i] = 0x00;
```

```
                }
```

```
                changeBytes(path, offset, data);
```

```
                for (int i = 0; i < data.Length; i++)
```

```
                {
```

```
                    tempfile[i] = data[i];
```

```

    }
    File.Delete(path);
    }
}

private static void changeBytes(string path, int offset, byte[] data)
{
    if (!File.Exists(path))
    {
        System.Windows.Forms.MessageBox.Show("File does not exist!");
        return;
    }

    using (Stream s = File.Open(path, FileMode.Open))
    {
        s.Position = offset;
        s.Write(data, 0, data.Length);
        System.Windows.Forms.MessageBox.Show("File permanently removed!");
    }
}
}

```

Dexcryptmin.cs

PHP Code

<?php

/\*

\* Copyright (C) 2011, Dextrey (0xDEADDEAD)

\* LINK: <https://hackforums.net/showthread.php?tid=1481595>

\*/

using System;

using System.Collections.Generic;

using System.Text;

class DexCryptMin

{

public static string Encrypt(string plain, string key)

{

return Convert.ToBase64String(Encrypt(UnicodeEncoding.UTF8.GetBytes(plain),  
UnicodeEncoding.UTF8.GetBytes(key)));

}

public static string Decrypt(string plain, string key)

{

return UnicodeEncoding.UTF8.GetString(Decrypt(Convert.FromBase64String(plain),  
UnicodeEncoding.UTF8.GetBytes(key)));

}

public static byte[] Encrypt(byte[] plain, byte[] key)

{

byte[] expandedKey = ExpandKey(key, plain.Length);

```

byte[] wholeState = plain;
byte magic = (byte)new Random().Next(byte.MaxValue);
Array.Resize(ref wholeState, plain.Length + 1);
wholeState[wholeState.Length - 1] = magic;

for (int i = 0; i < wholeState.Length - 1; i++)
{
    wholeState[i] = (byte)(wholeState[i] ^ expandedKey[i] ^ magic);
}

return wholeState;
}

public static byte[] Decrypt(byte[] plain, byte[] key)
{
    byte[] expandedKey = ExpandKey(key, plain.Length);
    byte[] wholeState = plain;
    byte magic = plain[plain.Length - 1];
    Array.Resize(ref wholeState, wholeState.Length - 1);

    for (int i = 0; i < wholeState.Length; i++)
    {
        wholeState[i] = (byte)(wholeState[i] ^ magic ^ expandedKey[i]);
    }

    return wholeState;
}

/// <summary>
/// Performs DexCryptMin key expansion algorithm on variable length input key
/// </summary>
/// <param name="key">Input key</param>
/// <param name="length">Count of output keystream bytes</param>
/// <returns>Expanded keystream</returns>
private static byte[] ExpandKey(byte[] key, int length)
{
    {
        if (key.Length >= length) return key;
        byte[] rconst = BitConverter.GetBytes(Math.Round(Math.PI, 3));
        byte[] result = new byte[length];
        Buffer.BlockCopy(key, 0, result, 0, key.Length);
        // init round (fill all remaining bytes)
        for (int i = key.Length; i < length; i++)
        {
            // x[i] = ((k[i - len(k)] % len(k)] + x[i - 1]) % 256
            result[i] = (byte)((key[(i - key.Length) % key.Length] ^ (result[i - 1])) % 256);
        }
        // main rounds (process all bytes)
        for (int round = 0; round < 2; round++)
        {
            result[0] = (byte)(result[0] ^ rconst[round]);
            for (int i = 1; i < result.Length; i++)
            {
                // x[i] = ((x[i] ^ (rcon[r] << (i % 3))) ^ x[i - 1]) % 256
                result[i] = (byte)((((result[i] ^ (byte)(rconst[round] << (i % 4))) ^ result[i - 1]) % 256);
            }
        }
    }
}

```

```

    }
    }
    return result;
    }
}

```

A secure way to remove your files.

**d3agle** - July 24, 2015, 11:13 pm

Nice source, but I'm a little confused why you are encrypting the temporary file in memory, and only writing zeros to disk... Kind of defeats the purpose of the encryption, right? :P

**Immortal** - July 25, 2015, 10:41 am

(Jul 24, 2015 11:13 PM)d3agle Wrote: [\[CITE\]](#) Nice source, but I'm a little confused why you are encrypting the temporary file in memory, and only writing zeros to disk... Kind of defeats the purpose of the encryption, right? :P

wanted to make sure that no trails of data were left behind for lets say a forensic data analysis team. That way if there were tiny bits in the ram that didnt get changed to zeros do to some kind of anomaly the data is encrypted. So I take the size of the file to a byte[] and then encrypt the one in the memory and overwrite with 0's to make sure it's gone then delete the file.

**d3agle** - July 25, 2015, 11:09 pm

(Jul 25, 2015 10:41 AM)Imm?rtal Wrote: [\[CITE\]](#) wanted to make sure that no trails of data were left behind for lets say a forensic data analysis team. That way if there were tiny bits in the ram that didnt get changed to zeros do to some kind of anomaly the data is encrypted. So I take the size of the file to a byte[] and then encrypt the one in the memory and overwrite with 0's to make sure it's gone then delete the file.

I see, thank you for explaining.

I rewrote a portion and played around with it a bit. The method for collecting the directory/file information is pretty hacky and could be done with a recursive method but I'm too lazy. Hell, the whole thing is hacky but w.e. lol.

I also made it so it overwrites the files with the encrypted bytes then again with the zeros.

Heres the source:

PHP Code

```

<?php
using System;
using System.Collections.Generic;
using System.IO;
using System.Text;
using System.Threading;
using System.Windows.Forms;

```

namespace Shredder

```

{
    /**
     * @Author: Immortal
     * @Modified by: d3agle
     */
    public partial class Form1 : Form
    {

```

```

private List<string> filePaths;
private List<string> directoryPaths;

private int filesShredded = 0;

public Form1()
{
    InitializeComponent();
    this.AllowDrop = true;
    this.DragEnter += new DragEventHandler(Form1_DragEnter);
    this.DragDrop += new DragEventHandler(Form1_DragDrop);
}

private void Form1_Load(object sender, EventArgs e)
{
    filePaths = new List<string>();
    directoryPaths = new List<string>();
}

private void Form1_DragEnter(object sender, DragEventArgs e)
{
    if (e.Data.GetDataPresent(DataFormats.FileDrop)) e.Effect = DragDropEffects.Copy;
}

private void Form1_DragDrop(object sender, DragEventArgs e)
{
    string[] paths = (string[]) e.Data.GetData(DataFormats.FileDrop);

    directoryPaths.Clear();
    filePaths.Clear();

    foreach (string path in paths)
    {
        SortFileTypes(path);
    }

    ShredFiles();
}

private void SortFileTypes(string path)
{
    FileAttributes attr = File.GetAttributes(path);

    if ((attr & FileAttributes.Directory) == FileAttributes.Directory)
    {
        string[] directories = Directory.GetDirectories(path, "*", SearchOption.AllDirectories);

        if (directories.Length > 0)
        {
            directoryPaths.Add(path);
            string[] tfiles = Directory.GetFiles(path, "*", SearchOption.TopDirectoryOnly);
            foreach (string file in tfiles)
            {

```

```

filePaths.Add(file);
}

foreach (string directoryPath in directories)
{
    directoryPaths.Add(directoryPath);

    string[] files = Directory.GetFiles(directoryPath, "*", SearchOption.TopDirectoryOnly);
    foreach (string file in files)
    {
        filePaths.Add(file);
    }
}
else
{
    directoryPaths.Add(path);

    string[] files = Directory.GetFiles(path, "*", SearchOption.TopDirectoryOnly);
    foreach (string file in files)
    {
        filePaths.Add(file);
    }
}
else
    filePaths.Add(path);
}

private void ShredFiles()
{
    new Thread(() =>
    {
        foreach (string filePath in filePaths)
        {
            ShredFile(filePath);
            filesShredded++;
        }

        directoryPaths.Reverse();
        foreach (string dirPath in directoryPaths)
        {
            Directory.Delete(dirPath);
            filesShredded++;
        }

        directoryPaths.Clear();
        filePaths.Clear();

        MessageBox.Show("Shredded Files Succesfully!");

    }).Start();
}

```

```

new Thread(() =>
{
int fileDirCount = (directoryPaths.Count + filePaths.Count);

while (filesShredded < fileDirCount)
{
decimal percent = filesShredded * 100 / fileDirCount;

this.Invoke((MethodInvoker) delegate
{
label1.Text = percent + "% Shredded";
});

Thread.Sleep(10);
}

this.Invoke((MethodInvoker) delegate
{
label1.Text = "100% Shredded";
});

}).Start();
}

private void ShredFile(string path)
{
FileAttributes attributes = File.GetAttributes(path);

if ((attributes & FileAttributes.ReadOnly) == FileAttributes.ReadOnly)
{
attributes = RemoveAttribute(attributes, FileAttributes.ReadOnly);
File.SetAttributes(path, attributes);
}

int offset = 0x0;
byte[] tempfile = File.ReadAllBytes(path);
byte[] data = new byte[tempfile.Length];
Random r = new Random();
byte[] key = new byte[256];
r.NextBytes(key);
tempfile = DexCryptMin.Encrypt(tempfile, key);

ChangeBytes(path, offset, tempfile);

for (int i = 0; i < data.Length; i++)
{
data[i] = 0x00;
}

ChangeBytes(path, offset, data);

File.Delete(path);
}

```

```

private void ChangeBytes(string path, int offset, byte[] data)
{
    if (!File.Exists(path))
        return;

    using (Stream s = File.Open(path, FileMode.Open))
    {
        s.Position = offset;
        s.Write(data, 0, data.Length);
    }
}

private FileAttributes RemoveAttribute(FileAttributes attributes,
FileAttributes attributesToRemove)
{
    return attributes & ~attributesToRemove;
}

/*
 * Copyright (C) 2011, Dextrey (0xDEADDEAD)
 * LINK: https://hackforums.net/showthread.php?tid=1481595
 */
public class DexCryptMin
{
    public static string Encrypt(string plain, string key)
    {
        return Convert.ToBase64String(Encrypt(UnicodeEncoding.UTF8.GetBytes(plain),
UnicodeEncoding.UTF8.GetBytes(key)));
    }

    public static string Decrypt(string plain, string key)
    {
        return UnicodeEncoding.UTF8.GetString(Decrypt(Convert.FromBase64String(plain),
UnicodeEncoding.UTF8.GetBytes(key)));
    }

    public static byte[] Encrypt(byte[] plain, byte[] key)
    {
        byte[] expandedKey = ExpandKey(key, plain.Length);
        byte[] wholeState = plain;
        byte magic = (byte) new Random().Next(byte.MaxValue);
        Array.Resize(ref wholeState, plain.Length + 1);
        wholeState[wholeState.Length - 1] = magic;

        for (int i = 0; i < wholeState.Length - 1; i++)
        {
            wholeState[i] = (byte) (wholeState[i] ^ expandedKey[i] ^ magic);
        }

        return wholeState;
    }
}

```

```

public static byte[] Decrypt(byte[] plain, byte[] key)
{
    byte[] expandedKey = ExpandKey(key, plain.Length);
    byte[] wholeState = plain;
    byte magic = plain[plain.Length - 1];
    Array.Resize(ref wholeState, wholeState.Length - 1);

    for (int i = 0; i < wholeState.Length; i++)
    {
        wholeState[i] = (byte) (wholeState[i] ^ magic ^ expandedKey[i]);
    }

    return wholeState;
}

/// <summary>
/// Performs DexCryptMin key expansion algorithm on variable length input key
/// </summary>
/// <param name="key">Input key</param>
/// <param name="length">Count of output keystream bytes</param>
/// <returns>Expanded keystream</returns>
private static byte[] ExpandKey(byte[] key, int length)
{
    if (key.Length >= length) return key;
    byte[] rconst = BitConverter.GetBytes(Math.Round(Math.PI, 3));
    byte[] result = new byte[length];
    Buffer.BlockCopy(key, 0, result, 0, key.Length);
    // init round (fill all remaining bytes)
    for (int i = key.Length; i < length; i++)
    {
        // x[i] = ((k[i - len(k)] % len(k)] + x[i - 1]) % 256
        result[i] = (byte) ((key[(i - key.Length) % key.Length] ^ (result[i - 1])) % 256);
    }
    // main rounds (process all bytes)
    for (int round = 0; round < 2; round++)
    {
        result[0] = (byte) (result[0] ^ rconst[round]);
        for (int i = 1; i < result.Length; i++)
        {
            // x[i] = ((x[i] ^ (rcon[r] << (i % 3))) ^ x[i - 1]) % 256
            result[i] = (byte) (((result[i] ^ (byte) (rconst[round] << (i % 4))) ^ result[i - 1]) % 256);
        }
    }
    return result;
}
}
}

```

Immortal - July 26, 2015, 10:33 am

(Jul 25, 2015 11:09 PM)d3agle Wrote: [\[CITE\]](#) I see, thank you for explaining.

I rewrote a portion and played around with it a bit. The method for collecting the directory/file

information is pretty hacky and could be done with a recursive method but I'm too lazy. Hell, the whole thing is hacky but w.e. lol.

I also made it so it overwrites the files with the encrypted bytes then again with the zeros.

Heres the source:

PHP Code

```
<?php
using System;
using System.Collections.Generic;
using System.IO;
using System.Text;
using System.Threading;
using System.Windows.Forms;

namespace Shredder
{
    /**
     * @Author: Immortal
     * @Modified by: d3agle
     */
    public partial class Form1 : Form
    {
        private List<string> filePaths;
        private List<string> directoryPaths;

        private int filesShredded = 0;

        public Form1()
        {
            InitializeComponent();
            this.AllowDrop = true;
            this.DragEnter += new DragEventHandler(Form1_DragEnter);
            this.DragDrop += new DragEventHandler(Form1_DragDrop);
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            filePaths = new List<string>();
            directoryPaths = new List<string>();
        }

        private void Form1_DragEnter(object sender, DragEventArgs e)
        {
            if (e.Data.GetDataPresent(DataFormats.FileDrop)) e.Effect = DragDropEffects.Copy;
        }

        private void Form1_DragDrop(object sender, DragEventArgs e)
        {
            string[] paths = (string[]) e.Data.GetData(DataFormats.FileDrop);

            directoryPaths.Clear();
            filePaths.Clear();
        }
    }
}
```

```

foreach (string path in paths)
{
    SortFileTypes(path);
}

ShredFiles();
}

private void SortFileTypes(string path)
{
    FileAttributes attr = File.GetAttributes(path);

    if ((attr & FileAttributes.Directory) == FileAttributes.Directory)
    {
        string[] directories = Directory.GetDirectories(path, "*", SearchOption.AllDirectories);

        if (directories.Length > 0)
        {
            directoryPaths.Add(path);
            string[] tfiles = Directory.GetFiles(path, "*", SearchOption.TopDirectoryOnly);
            foreach (string file in tfiles)
            {
                filePaths.Add(file);
            }

            foreach (string directoryPath in directories)
            {
                directoryPaths.Add(directoryPath);

                string[] files = Directory.GetFiles(directoryPath, "*", SearchOption.TopDirectoryOnly);
                foreach (string file in files)
                {
                    filePaths.Add(file);
                }
            }
        }
        else
        {
            directoryPaths.Add(path);

            string[] files = Directory.GetFiles(path, "*", SearchOption.TopDirectoryOnly);
            foreach (string file in files)
            {
                filePaths.Add(file);
            }
        }
        else
        {
            filePaths.Add(path);
        }

        private void ShredFiles()
        {

```

```

new Thread() =>
{
foreach (string filePath in filePaths)
{
ShredFile(filePath);
filesShredded++;
}

directoryPaths.Reverse();
foreach (string dirPath in directoryPaths)
{
Directory.Delete(dirPath);
filesShredded++;
}

directoryPaths.Clear();
filePaths.Clear();

MessageBox.Show("Shredded Files Succesfully!");

}).Start();

new Thread() =>
{
int fileDirCount = (directoryPaths.Count + filePaths.Count);

while (filesShredded < fileDirCount)
{
decimal percent = filesShredded * 100 / fileDirCount;

this.Invoke((MethodInvoker) delegate
{
label1.Text = percent + "% Shredded";
});

Thread.Sleep(10);
}

this.Invoke((MethodInvoker) delegate
{
label1.Text = "100% Shredded";
});

}).Start();
}

private void ShredFile(string path)
{
FileAttributes attributes = File.GetAttributes(path);

if ((attributes & FileAttributes.ReadOnly) == FileAttributes.ReadOnly)
{
attributes = RemoveAttribute(attributes, FileAttributes.ReadOnly);
}
}

```

```

File.SetAttributes(path, attributes);
}

int offset = 0x0;
byte[] tempfile = File.ReadAllBytes(path);
byte[] data = new byte[tempfile.Length];
Random r = new Random();
byte[] key = new byte[256];
r.NextBytes(key);
tempfile = DexCryptMin.Encrypt(tempfile, key);

ChangeBytes(path, offset, tempfile);

for (int i = 0; i < data.Length; i++)
{
    data[i] = 0x00;
}

ChangeBytes(path, offset, data);

File.Delete(path);
}

private void ChangeBytes(string path, int offset, byte[] data)
{
    if (!File.Exists(path))
        return;

    using (Stream s = File.Open(path, FileMode.Open))
    {
        s.Position = offset;
        s.Write(data, 0, data.Length);
    }
}

private FileAttributes RemoveAttribute(FileAttributes attributes,
FileAttributes attributesToRemove)
{
    {
        return attributes & ~attributesToRemove;
    }
}

/*
 * Copyright (C) 2011, Dextrey (0xDEADDEAD)
 * LINK: https://hackforums.net/showthread.php?tid=1481595
 */
public class DexCryptMin
{
    public static string Encrypt(string plain, string key)
    {
        return Convert.ToBase64String(Encrypt(UnicodeEncoding.UTF8.GetBytes(plain),
UnicodeEncoding.UTF8.GetBytes(key)));
    }
}

```

```

public static string Decrypt(string plain, string key)
{
    return UnicodeEncoding.UTF8.GetString(Decrypt(Convert.FromBase64String(plain),
UnicodeEncoding.UTF8.GetBytes(key)));
}

```

```

public static byte[] Encrypt(byte[] plain, byte[] key)
{
    byte[] expandedKey = ExpandKey(key, plain.Length);
    byte[] wholeState = plain;
    byte magic = (byte) new Random().Next(byte.MaxValue);
    Array.Resize(ref wholeState, plain.Length + 1);
    wholeState[wholeState.Length - 1] = magic;

    for (int i = 0; i < wholeState.Length - 1; i++)
    {
        wholeState[i] = (byte) (wholeState[i] ^ expandedKey[i] ^ magic);
    }

    return wholeState;
}

```

```

public static byte[] Decrypt(byte[] plain, byte[] key)
{
    byte[] expandedKey = ExpandKey(key, plain.Length);
    byte[] wholeState = plain;
    byte magic = plain[plain.Length - 1];
    Array.Resize(ref wholeState, wholeState.Length - 1);

    for (int i = 0; i < wholeState.Length; i++)
    {
        wholeState[i] = (byte) (wholeState[i] ^ magic ^ expandedKey[i]);
    }

    return wholeState;
}

```

```

/// <summary>
/// Performs DexCryptMin key expansion algorithm on variable length input key
/// </summary>
/// <param name="key">Input key</param>
/// <param name="length">Count of output keystream bytes</param>
/// <returns>Expanded keystream</returns>
private static byte[] ExpandKey(byte[] key, int length)
{
    if (key.Length >= length) return key;
    byte[] rconst = BitConverter.GetBytes(Math.Round(Math.PI, 3));
    byte[] result = new byte[length];
    Buffer.BlockCopy(key, 0, result, 0, key.Length);
    // init round (fill all remaining bytes)
    for (int i = key.Length; i < length; i++)
    {
        // x[i] = ((k[i - len(k)] % len(k)] + x[i - 1]) % 256

```

```

    result[i] = (byte) ((key[(i - key.Length) % key.Length] ^ (result[i - 1])) % 256);
}
// main rounds (process all bytes)
for (int round = 0; round < 2; round++)
{
    result[0] = (byte) (result[0] ^ rconst[round]);
    for (int i = 1; i < result.Length; i++)
    {
        // x[i] = ((x[i] ^ (rcon[r] << (i % 3))) ^ x [i - 1]) % 256
        result[i] = (byte) (((result[i] ^ (byte) (rconst[round] << (i % 4))) ^ result[i - 1]) % 256);
    }
}
return result;
}
}
}

```

Hmmm, thats weird I could not get the application to accept drag and drop even allowed it in the properties. it was confusing me maybe i missed something.