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

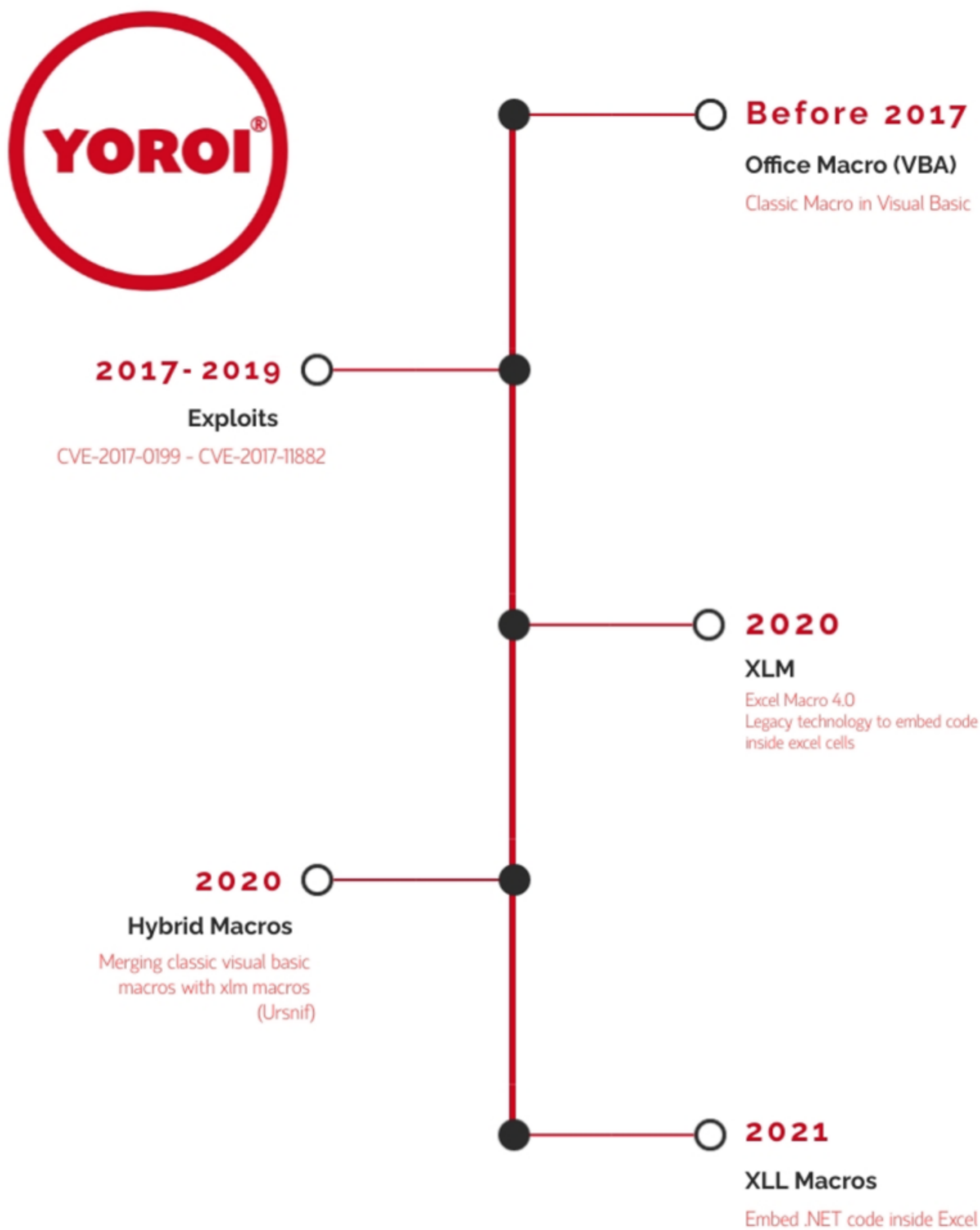
- CVE-2017-11882 is a remote code execution vulnerability allowing the attacker to execute a shellcode embedded inside the malicious document, due to a flaw in memory handling of the Equation Editor component, present inside of all Office applications. We tracked this technique in many reports, and we 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:

Hash	994013d66ae20cfa4ef1097d73481b00a672131d0de44d79a04ff12f492aae55
Threat	XLL Dropper
Brief Description	Malicious XLL file, dropping several payloads
SSDEEP	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.

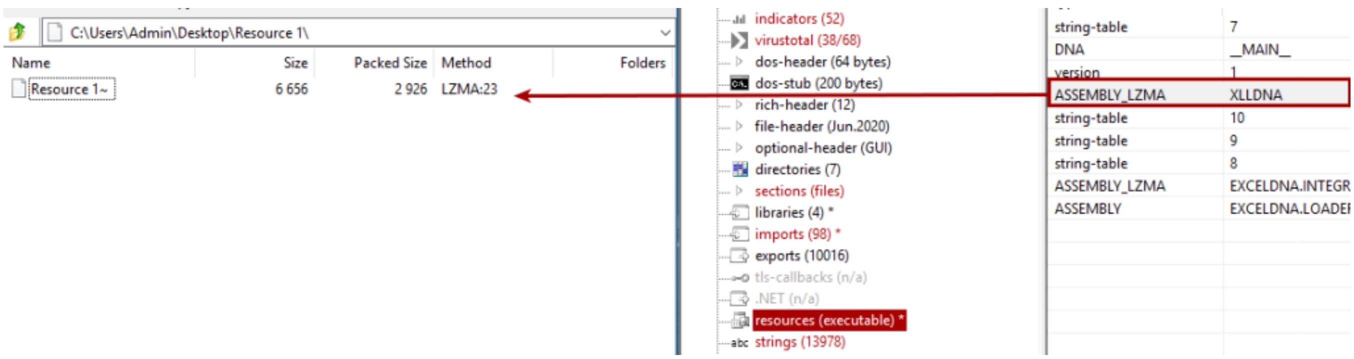


Figure 1: Static information of the EXCEL-DNA component and relative manual extraction

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.

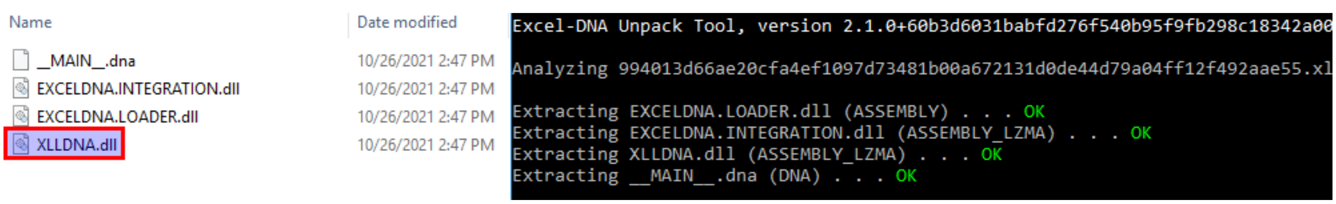


Figure 2: Extraction of .xll file

The DLL payload

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

Hash	8f9dcf822dd8f22dd3c21f0798e97554a24b05a0fa3065d2580933ff4af29a6d
Threat	.NET dll embedded payload
Brief Description	Payload contained inside the XLL file.
SSDEEP	96:mFCZXPFomsKQrdLVaBlP1WiGxB7BHjA5ASDBmq9:mFClvKQrnanQ39HjA2on

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:

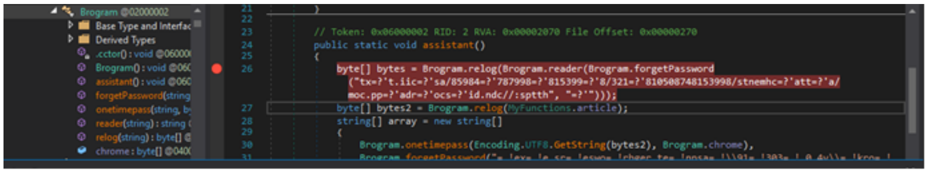


Figure 3: Decoding the first DropURL

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.

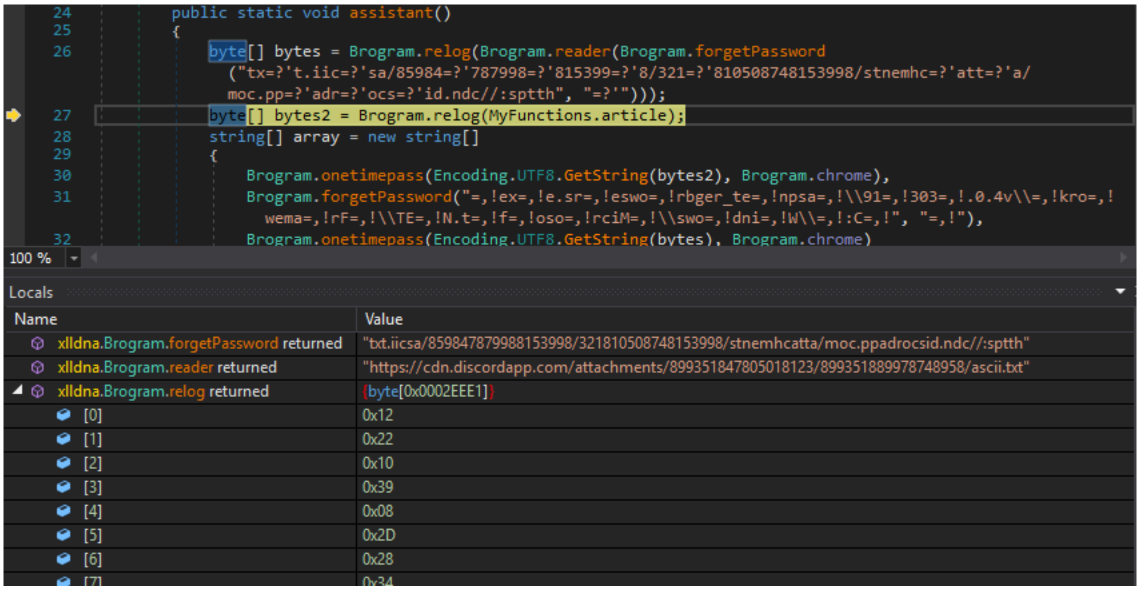


Figure 4: First Encrypted Payload

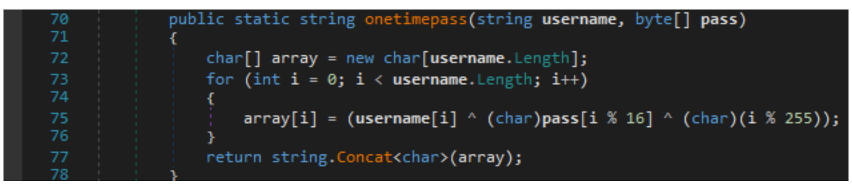


Figure 5: XOR Decryption Function

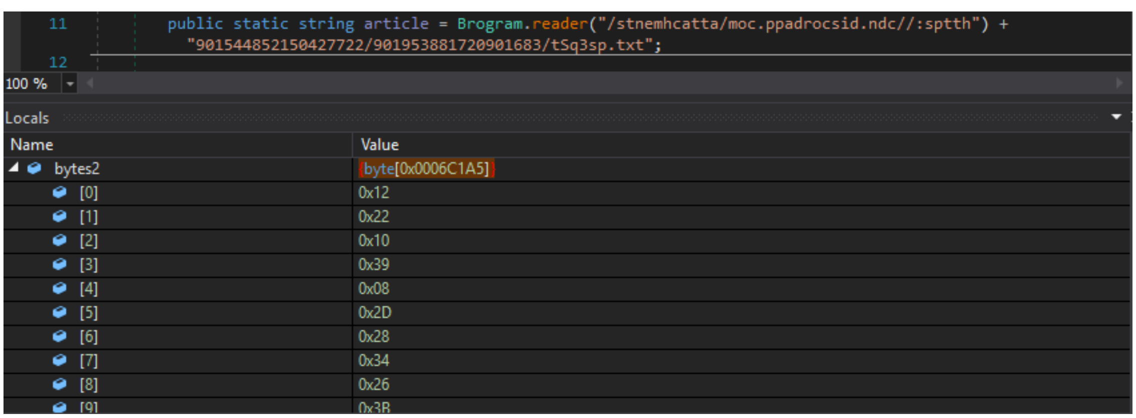


Figure 6: Second Encrypted Payload

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.

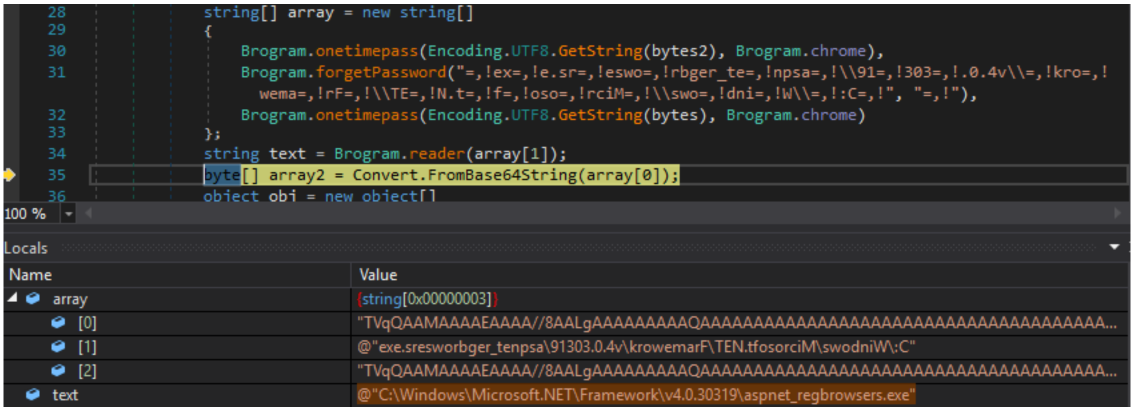


Figure 7: Decoding the payloads

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 **aspnet_regbrowsers.exe**
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.

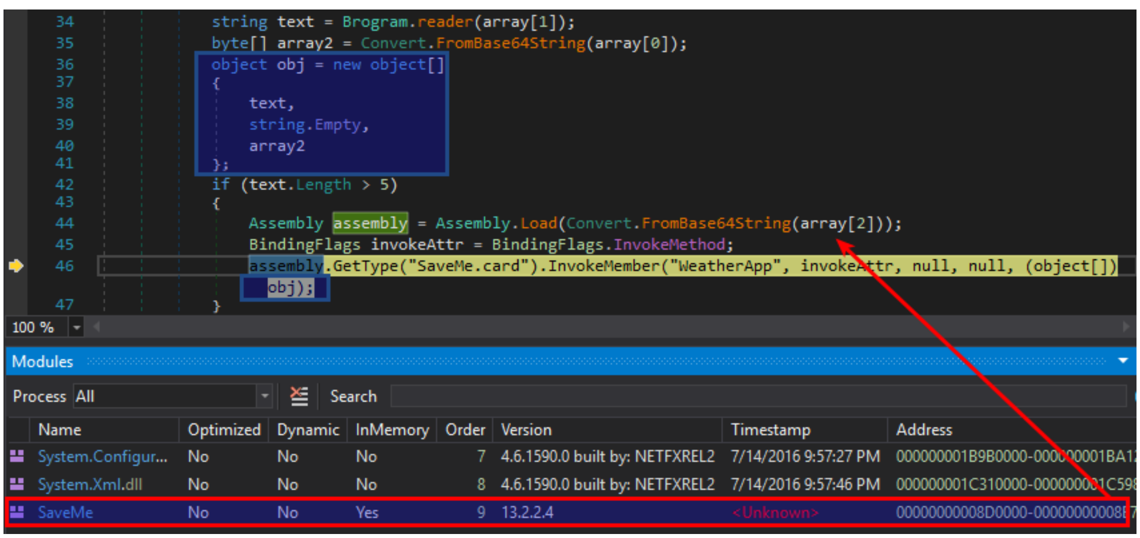


Figure 8: Payload loaded in memory

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.

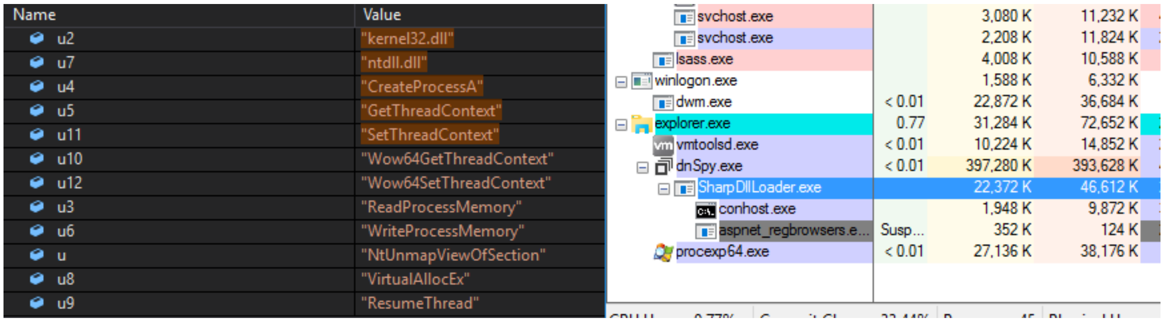


Figure 9: Classic Process Hollowing into aspnet_regbrowsers.exe

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.

Hash	50d645e57a915baf4db98b6476681dce65d809e84f2c72eff0d6db4b10fd28d0
Threat	AgentTesla Stealer
Brief Description	Obfuscated AgentTesla
SSDEEP	3072:Q9Wgl88xlaXntoTAKeNGUsE1M+IJkE0oU6btrJ58low2wefpxSqL8cQWxQq8E3zH:QzVtok0UY+qkR298lrmv4HWsE3z6UJ

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.

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:

Hash	64a668add3d7f3bbcc0ef6acb25529c70df773d74e7e17a4a8fd8c95e81ee8bd
Threat	Formbook
Brief Description	Formbook payload retrieved in a second time from the dropurl
SSDEEP	3072:W7psS2npp9ymO/pw4imY0bXkN6edhTDYEUvCJ6Trad+:Wu/emIpwdrTN6edhvYdg6fR

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.

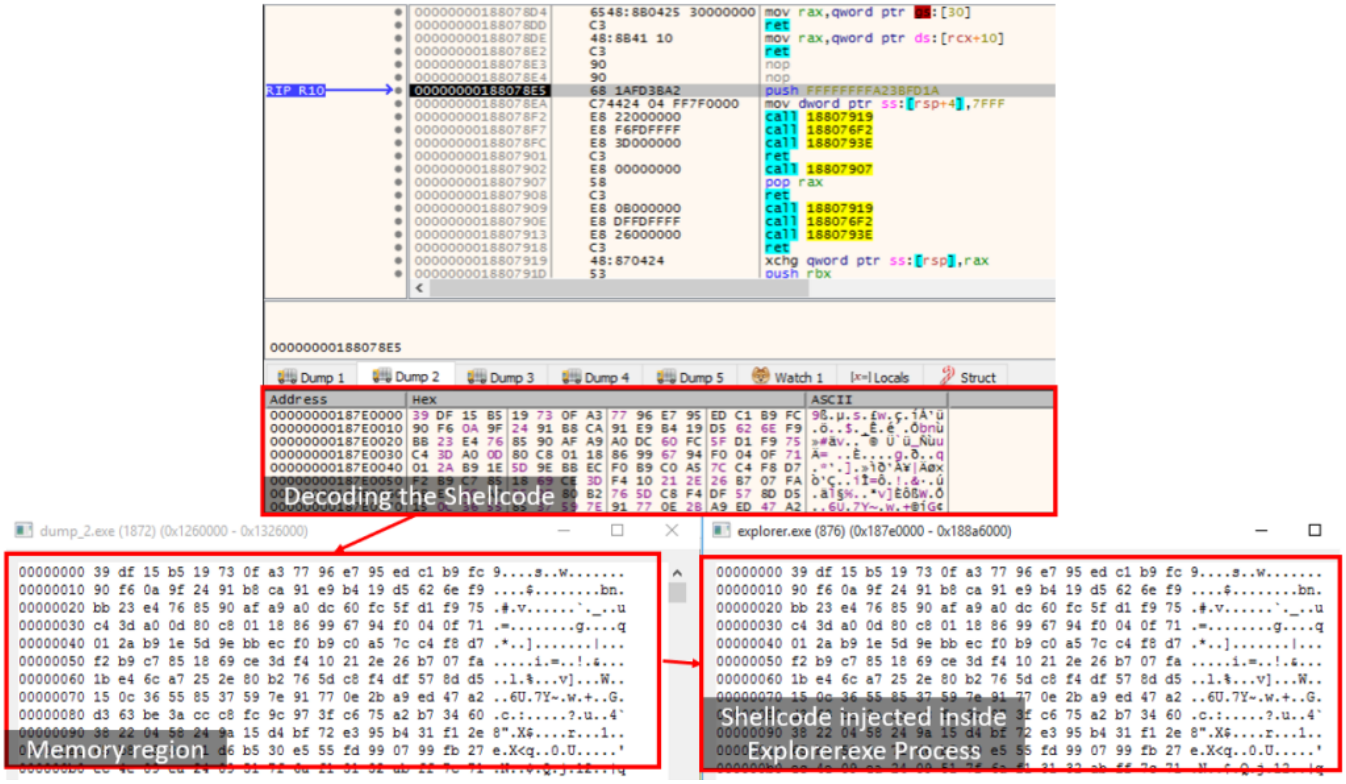


Figure 12: Shellcode injected in explorer routine

Hash	7f1f224a14a2e412a8c22535fc584c31bbcfe41241eb794c605c91987996d62e
Threat	Dridex
Brief Description	Dridex dropper
SSDEEP	768:ceQJmg+fxfveZ5RI3dO1+IpwY5xW04HPJ4hLqm9NdUPhnutmbX+NFw2WP0t9gE53:6f+f9eZzx++5SHhQ+qTciMIgAmw

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.

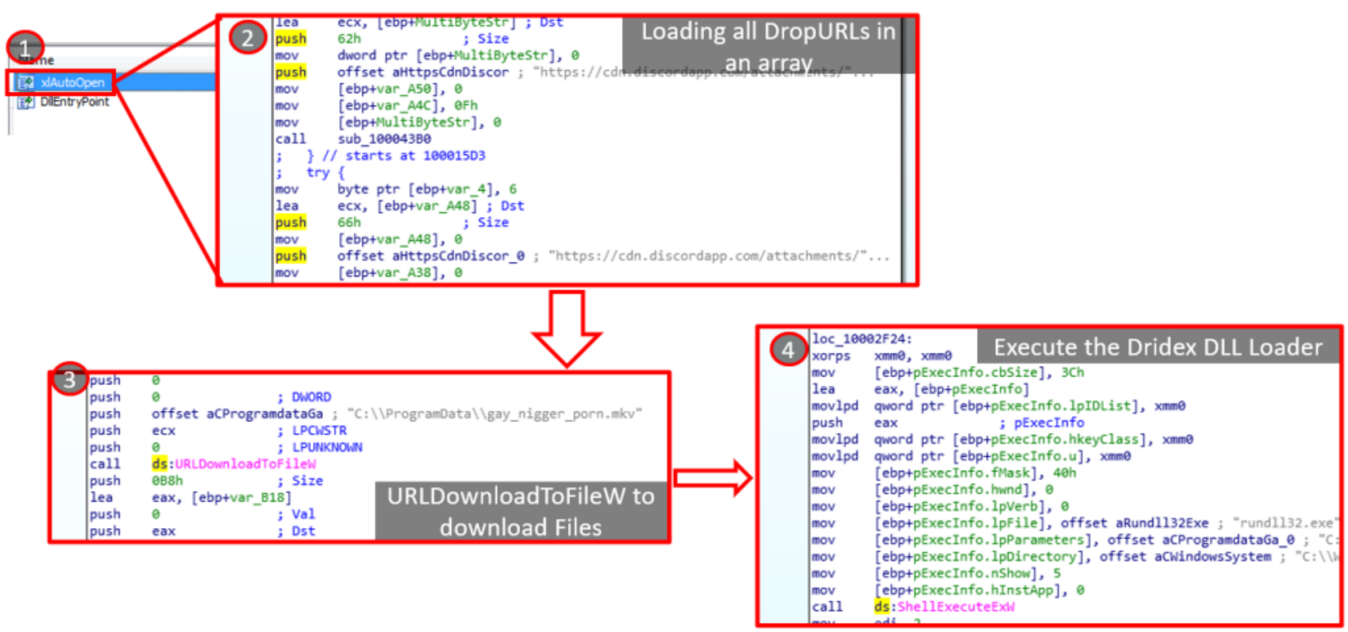


Figure 13: Dridex XLL dropper

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
- 2bebba83d0caec961116d39f9f52dbb2277c937ceef88326b34b646de3763fd0

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.