Analysis of August stealer malware

b hazmalware.blogspot.de/2016/12/analysis-of-august-stealer-malware.html

August	Statistics	Storage	Logs	Settings								Logout
Aug	ust											
Username:	:						Do	cument extensions (u	use ' ' as	s seperator):		
								*.txt *.doc				
Password:							Ma	Max. file size for upload (bytes):				
-								2000000				
Allowed re	port types:						Dir	ectory for search:				
Passwords	Wallets	Rdp Files	IM Clier	ts FTP Cl	ents Documen	ts Cookies		Drives (C:/, D;/ etc.))	Desktop	Documents	
							Sea	arch type:				
Allow requests by user who already is in logs					Search files only in top directory							
	Save changes					Delete all reports and logs						

NOTE: This blog has been merged with WordPress. You will be redirected to this article on the WordPress site in 10 seconds...

If you want to go there now, click here -<u>https://hazmalware.wordpress.com/2016/12/27/analysis-of-august-stealer-</u> <u>malware/</u>

August malware is designed to steal various data from compromised systems. It was observed appearing for sale around 10/20/2016. According to the malware authors post on underground forums it has the ability to steal various passwords, cookies, bitcoin wallets, RDP and FTP saved connections, and can even grab specified files. At the time of this writing the latest version can steal data from the following applications:

Browsers:

Mozilla FireFox Google Chrome Comodo IceDragon Vivaldi Browser Mail.Ru Browser Torch Browser Dooble Browser U Browser Coowon Amigo Browser Bromium Yandex Browser Opera Browser Chromium SRWare Iron CoolNovo Browser RockMelt Browser

FTP Clients:

FileZilla CoreFTP CuteFTP SmartFTP WinSCP Total Commander

Email Clients:

MS Outlook <= 2013

Mozilla Thunderbird

IM Clients:

Windows Live Pidgin Psi

Bitcoin Wallets:

wallet.dat

RDP remote connection files

Any specified files/documents

Here is a look at the admin panel of this malware:

August

Username:	Document extensions (use ' ' as seperator):			
-	*Ant *.doc			
Password:	Max. file size for upload (bytes):			
and as	2000000			
Allowed report types:	Directory for search:			
Passwords Wallets Rdp Files IM Clients PTP Clients Decuments Cookies	Drives (Ci/, Dj/ etc.) Desktop Documents			
	Search type:			
Allow requests by user who already is in logs	Search files only in top directory			
Save changes	Delete all reports and logs			

The malware has been observed being delivered in malspam emails that have word document attachments. It is the typical phishing type email that entices the user to open the attachment. The word documents contain VBA that will use powershell to download the malware payload from a remote URL where it is encoded into a byte array in the webpage. It is then programmed to load the byte array and decode it with an exclusive or (XOR) and call the main function from the loaded executable.

, 0x5b, 0x5b,

This post is geared more towards the binary analysis of the August malware, but the great people over at Proofpoint have done a fantastic job on an overview of the threat - you can read it on their blog <u>here</u>. So let's continue with our analysis.

Running the strings command on files can often give you quite a bit of interesting information regarding the sample you are analyzing. You have to be careful though, because strings can sometimes be misleading as malware authors can put junk strings in to confuse analysis -- remember to validate anything you find. Here are a few of the interesting strings I found in this sample:

FromBase64String ToBase64String NSSBase64Ptr_DecodeBuffer **IsDebuggerPresent** OutputDebugString FailFast Debugger get_IsAttached IsLogging get_IsAlive ConfusedByAttribute Confuser v1.9.0.0 HttpWebResponse HttpWebRequest CreateDecryptor ICryptoTransform crypt32.dll CryptoStream CryptoStreamMode Encrypt Decrypt PK11Ptr_Decrypt CryptEncrypt CryptDecrypt DOMAIN PASSWORD DOMAIN_VISIBLE_PASSWORD NtSetInformationProcess

You can see that there appears to be some strings related to crypto functions, encryption, decryption, HTTP functions, and even some anti-debugging. There are many, many more but we have to consolidate for the post. One of the more interesting items is the strings 'ConfusedByAttribute' and 'Confuser v1.9.0.0'.

Confuser is a packer / obfuscation tool for .NET applications. It offers a variety of obfuscation methods such as anti-debugging, anti-memory dumping, anti-decompiling, encrypting constants, methods, and resources, etc.

When analyzing an unknown binary it is always good to verify what type of file you are dealing with. PEiD shows that this is a .NET executable designed for 32 bit computers.

🦉 PEiD v0.	95							
File: C:\Us	sers \ab \Desktop \august.mal							
Entrypoint:	0001A83E	EP Section: .text >						
File Offset:	00018A3E	First Bytes: FF,25,00,20 >						
Linker Info:	8.0	Subsystem: Win32 console >						
Microsoft Vi	sual C# / Basic .NET							
Multi Scan	Multi Scan Task Viewer Options About Exit							
Stay on t	top	>>						

DiE (detect it easy) is another good tool to analyze exe files, especially if you think it might be packed. Looking at our sample it shows again that this is a .NET executable and it was designed for 32 bit. It also shows that this sample was packed with Confuser v1.9.0.0 - just like we saw in our strings.

Detect It Eas	sy 1.01	_ 🗆 🗵
File name:	C:/Users/lab/Desktop/august.mal	
Scan Scrip	ts Plugins Log	_
Type:	PE Plugins(Alt+3) 12912 Entropy FLC S H	
Export	Import Resource Overlay .NET PE	
EntryPoint:	0001a83e > ImageBase: 00400000	
NumberOfSect	ions: 0003 > SizeOfImage: 00020000	
protector	Confuser(1.9.0.0)[-] ?	
library	.NET(v2.0.50727)[-] ?	Options
linker	Microsoft Linker(8.0)[DLL32,console] ?	About
	Signatures 265 ms Scan	Exit

It is good practice to not just rely on a single source of information, but to verify with multiple sources. CFF Explorer is one of my favorite apps for analyzing binary files because it gives you so much detailed information. Looking at this in CFF shows much of the same.

CFF Explorer VIII - [august.mal] File Settings ?						
🔗 🤳 🖍	august.mal					
	Property	Value				
📮 🛅 File: august.mal	File Name	C:\User	s\ab\Desktop\august.mal			
Dos Header Int Headers	File Type	Portable	e Executable 32 .NET Assembly			
III Nt Headers III File Header	File Info	Microso	ft Visual Studio .NET			
Optional Header	File Size	100.50	KB (102912 bytes)			
Data Directories [x]	PE Size	100.50	KB (102912 bytes)			
— I Section Headers [x]	Created		v 27 December 2016, 14.40.41			
- Contractory - Contractory	Modified					
Resource Directory	Accessed	-	Fuesday 27 December 2016, 14:40.41			
Contraction Directory Contraction Directory Contraction			445C509E6A86EDE366076502E085A			
MetaData Header						
E I MetaData Streams	SHA-1	47ED88	47EDB81B7AB25A632C6D2911581E800F700208B2			
Tables Header	Property		Value			
🖵 🕼 Tables	Comments					
#Strings	CompanyName					
	FileDescription	August@				
#Blob	FileVersion		1.0.0.0			
	InternalName		August@.dl			
— 🐁 Dependency Walker	LegalCopyright LegalTrademarks		Copyright © 2016			
- Mex Editor						
- 🀁 Identifier	OriginalFilenam	e	August@.dl			
	ProductName	_	August@			
Quick Disassembler M Rebuilder	ProductVersion		Auguster 1.0.0.0			
A.D. E.D.			Lo. D.			

We see that the File Type is listed as a Portable Executable 32 .NET Assembly and File Info shows Microsoft Visual Studio .NET. So we are probably dealing with a .NET file... but let's do a little more analysis just to make sure. Do not pay attention to the created/modified/access times as these are when this binary was copied to the windows analysis computer... remember windows MAC time rules.

The Dos Header shows the 4D5A MZ magic number indicating that this is an executable file.

ight 🔊 📕 🔊	august.mal						
	Member	Offset	Size	Value			
📮 🖬 File: august.mal	e_magic	0000000	Word	5A4D			
- I Dos Header	e_cblp	00000002	Word	0090			
Nt Headers	e_cp	00000004	Word	0003			

Checking the Optional Header info we can see the 2 byte value 010B (PE32) indicating that it is designed as a 32 bit application.

🛩 CFF Explorer VIII - [August@.dll]							×		
File Settings ?									
🤌 🜷 👘		August@.dll							
7	1.	Member	Offset	Size	Value	Meaning	^		
File: August@.dll Bos Header	ĥ	Magic	0000098	Word	010B	PE32			
- I I Nt Headers		MajorLinkerVersion	A6000000	Byte	08				
Ile Header Ile Optional Header		MinorLinkerVersion	000009B	Byte	00				
Data Directories [x]		SizeOfCode	000009C	Dword	00015E00		_		
- Section Headers [x]		SizeOfficializedData	000000.00	Durard	00000600				

In the Data Directories section, under NT Headers -> Optional Header, we can see that .NET MetaData Directory RVA & Size on the right details page both contain values. These are good indications that we are indeed dealing with a .NET executable.

.NET MetaData Directory RVA	00000168	Dword	00002008	.text
.NET MetaData Directory Size	0000016C	Dword	00000048	

One last thing to verify - .NET files only have 1 import and 1 function imported. Here we see mscoree.dll and _CorDllMain, respectfully:

🄌 🤳 👘	august.mal									
	Module Name		Imports		OFTs		TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)
🗆 🖻 File: august.mal	00018A2E		N/A		000189EC		000189F0	000189F4	000189F8	000189FC
Dos Header At Headers	szAnsi		(nFunctions)		Dword		Dword Dword		Dword	Dword
File Header	mscoree.dll		1	1 000		0001A814 00000000		00000000	0001A82E	00002000
Optional Header										
Data Directories [x]	OFTs FTs (IA)		T)	Hint Name						
Section Headers [x] Dimport Directory										
Resource Directory	Dword	Dword		Word		szAnsi				
	0001A820	0001A8	20	0000		_CorDIN	1ain			
- C Directory										
MetaData Header										
MetaData Streams										

We can now confidently say that we are dealing with a .NET executable file.

There are a couple of other items of interest that we will take a look at. The exe compile datetime that shows the date and time the project was compiled from VisualStudio. This can easily be changed by anyone with a little knowledge. The compile time for this binary shows the following

i 💫 📕 🔊	august.mal						
	Member	Offset	Size	Value	Meaning		
📮 🛅 File: august.mal	Machine	00000084	Word	014C	Intel 386		
Dos Header Int Headers	NumberOfSections	0000086	Word	0003			
	TimeDateStamp	00000088	Dword	5829ABBA			
Diffional Header	PointerToSymbolTable	000008C	Dword	00000000			
Data Directories [x]	NumberOfSymbols	00000090	Dword	00000000			
Section Headers [x]	SizeOfOptionalHeader	00000094	Word	00E0			
Import Directory Resource Directory	Characteristics	00000096	Word	2102	Click here		
Which converted becomes							

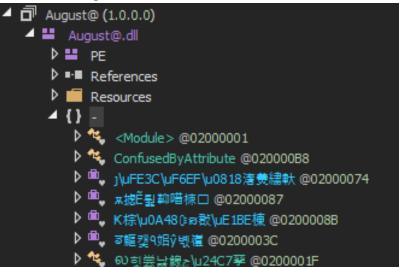
0x5829ABBA [Mon Nov 14 12:19:06 2016 UTC]

Given other indicators going on for this around the same time, confidence is pretty high that the compile time is legit. The other item of interest is the module directory that shows the

name of this file at compile time. Because anyone can rename a file at anytime, this entry will often show the original name of the file the malware author had set at the time it was compiled. Here we see it was indeed called 'August@.dll'

august.mal					
Module (1)	Member	Offset	Size	Value	Meaning
	Generation	0000B870	Word	0000	
I - TypeDef (184)	Name	0000BB72	Word	09C1	August@.dll
E- Field (236)	Mvid	0000BB74	Word	0001	GUID Index

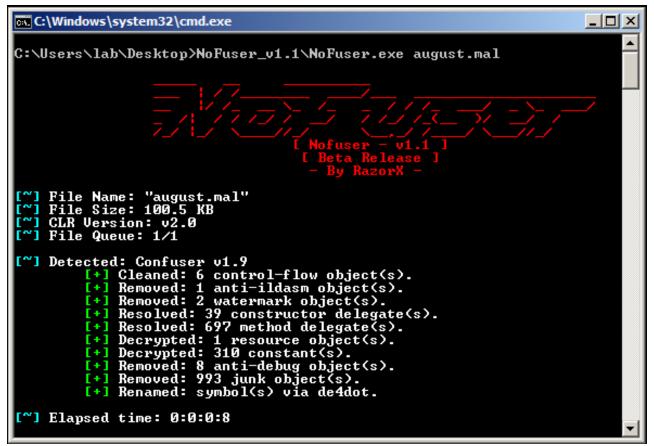
Now that we know we are dealing with a .NET file, we can open it up in a decompiler and see what we get.



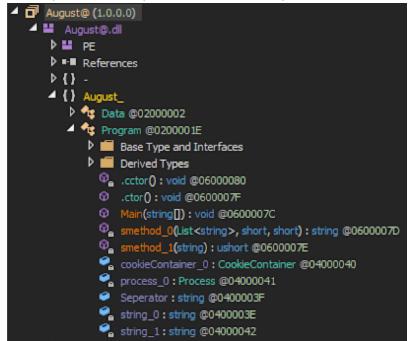
All of the methods and assemblies are encrypted by the Confuser packer that was identified earlier.



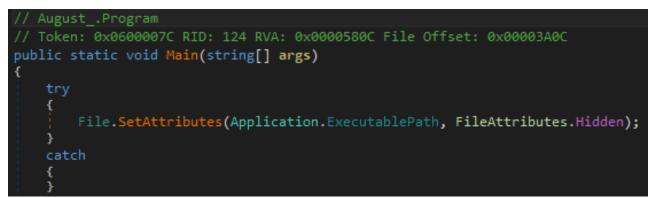
Looks like we're going to have to decrypt it first before analyzing. For this we can use a program called NoFuser to help with decryption. Running our binary through NoFuser shows that it detected Confuser v1.9 as well and successfully cleaned our binary.



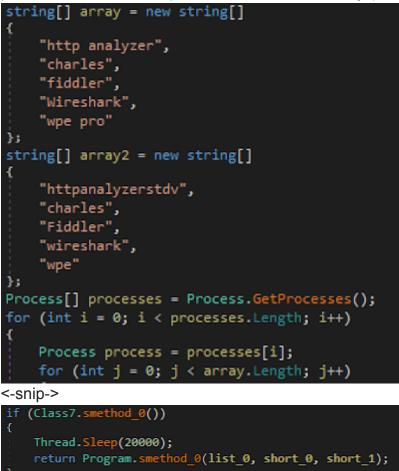
Now open it back up in side the decompiler and we should have readable data.



Moving into the Main() method, because this is where code execution starts, we see that right away the malware hides itself by setting the FileAttribute to 'Hidden'.



Next it has functions to check whether there are analysis programs currently running as processes, and will sleep for 20000ms if it finds any (roughly 30 seconds).



It then gathers some information about the computer that it has just infected, such as the type of CPU, amount of RAM, networking info, etc.

```
"AUG -% 0: CPU[",
Class9.smethod_3(),
"] BASE[",
Class9.smethod_8(),
"] BIOS[",
Class9.smethod_9(),
"]"
```

It also grabs the username of the person currently logged in



It then encrypts the data and sends it to the pre-programmed C2 server via the following web request



We can see that it is using a POST method and has a timeout value set to 100000ms. The C2 URL value is completely configurable and will change from sample to sample.

After checking in with its C2 the malware immediately begins its data exfiltration routines. Checking through the code shows functions for all the data exfil types listed at the beginning of this post. I will not list all of the code here for brevity sake.



All-in-all this is a pretty interesting sample. I am still analyzing and learning more about it, but I think that about wraps it up for this post!

The sample analyzed in this post was found on hybrid-analysis over here.

Filetype PE32 executable (DLL) (console) Intel 80386 Mono/.Net assembly, for MS Windows

Mimetype	application/x-dosexec
Size	102912
MD5	11c445c509f6a86fde366076502f085a
SHA1	47edb81b7ab25a632c6d2911581e800f700208b2
SHA256	c725e62b5aa3dfbf41b979bb55b04d43fa7042ca34cb914892872267e79de8d1