Taking the FIRST look at Crypt0l0cker

blog.talosintelligence.com/2017/08/first-look-crypt0l0cker.html

Indicators	Netw	ork	Processes	Artifacts	Registry
Size: 398357	Exports: 0	AV Sigs: 1		MD5: ba678704162934e5b	3d2f2c3b5e2168d
Artifact 3: 🖂 \Users	s\Administrator\AppData	\Local\Temp\nsr2B	42.tmp\System.dll		
Source: disk	Imports: 19	Type: DLL - P	E32 executable (DLL)	SHA256: 75ed40311875312	2617d6711baed0b
Size: 11264	Exports: 8	AV Sigs: 0		MD5: a436db0c473a087eb6	61ff5c53c34ba27
Artifact 4: 🖂 \Prog	ramData\uwupefovygigyl	lih\ebiwewiz			
Source: disk	Imports: 0	Type: data		SHA256: 59cc880a112b903	32e86f083bf83d3d
Size: 64	Exports: 0	AV Sigs: 0		MD5: 7cf5d4223f87fd146e6	52c4de65ffa13
Artifact 5: 🖂 \Prog	ramData\uwupefovygigyl	lih\ewiwobiz			
Source: disk	Imports: 0	Type: data		SHA256: f716a869683941d	23dd1cda790267
Size: 16	Exports: 0	AV Sigs: 0		MD5: 3db7df4f5bb88c4b655	526bad192a3e96
Artifact 6: 🖴 \Prog	ramData\uwupefovygigyl	lih\asiwahiz			
O a suma a suddata	Incomentary O				

This post is authored by <u>Matthew Molyett</u>.

Executive Summary

In March, Talos reported on the details of <u>Crypt0l0cker</u> based on an extensive analysis I carried out on the sample binaries. Binaries -- plural -- because, as noted in the original blog, the Crypt0l0cker payload leveraged numerous executable files which shared the same codebase. Those executables had nearly identical functions in each, but identifying all of those functions repeatedly is tedious and draws time away from improving the analysis. Enter <u>FIRST</u>, the Function Identification and Recovery Signature Tool released by Talos in December 2016.

<u>FIRST</u> allowed me to port my analysis from the unpacking dll to the payload file instantly. Once I was satisfied my analysis across both files, I was then handed a suspected previous version of the sample. <u>FIRST</u> was able to identify similar code across the versions and partially port the analysis back to the older file. When the next version of Crypt0l0cker comes out, I will be able to get a jump on my analysis by using FIRST to port that work forward to the similar code. You can use it to port my work to your sample as well. I will demonstrate doing just that with a Crypt0l0cker sample which appeared on VirusTotal in April 2017, more than a month after the Talos blog about it. There has been no targeted analysis of this file to provide background for this post.

Locating the Sample

Procuring a malware sample of a known family without analyzing it can feel like a heavy challenge to overcome. Thankfully, Talos can leverage Threat Grid sandbox reports of suspected malware samples that we receive. Such reports can be scanned for family IOCs. Per our previous analysis into Crypt0l0cker, the infection status of that version is stored in a file named ewiwobiz. By searching <u>Cisco Threat Grid</u> telemetry for files which created ewiwobiz, I identified a file which was probably a Crypt0l0cker executable.

Threat Grid	💠 Submit sample	A Indicators Q Search -	
Indicators	Net	work Processe	s Artifacts Regis
Size: 398357	Exports: 0	AV Sigs: 1	MD5: ba678704162934e5b3d2f2c3b5e216
Artifact 3: 🖂 \Users	Administrator\AppDa	ta\Local\Temp\nsr2B42.tmp\System	.dll
Source: disk	Imports: 19	Type: DLL - PE32 executable	(DLL) SHA256: 75ed40311875312617d6711bae
Size: 11264	Exports: 8	AV Sigs: 0	MD5: a436db0c473a087eb61ff5c53c34ba
Artifact 4: 🖂 \Progra	amData\uwupefovygig	gylih\ebiwewiz	
Source: disk	Imports: 0	Type: data	SHA256: 59cc880a112b9032e86f083bf83
Size: 64	Exports: 0	AV Sigs: 0	MD5: 7cf5d4223f87fd146e652c4de65ffa13
Artifact 5: 🖂 \Progra	amData\uwupefovygiş	gylih\ewiwobiz	
Source: disk	Imports: 0	Type: data	SHA256: 1716a869683941d23dd1cda7902
Size: 16	Exports: 0	AV Sigs: 0	MD5: 3db7df4f5bb88c4b65526bad192a3e
Artifact 6: 🖂 \Progra	amData\uwupefovygig	gylih\asiwahiz	
Source: disk	Imports: 0	Type: data	SHA256: 6fc2882e3707201708bd8ba76be
Size: 398368	Exports: 0	AV Sigs: 0	MD5: 8646fdc0803627d99dd2f3105e1ab0

With a report to investigate, I needed to procure the actual sample. My sandbox report shows that the suspected Crypt0l0cker file is nearly 400 kb and likely a <u>Nullsoft Installer</u> file, which is a common packager. Static file information gives me the file hash which arms me with the ability to continue my investigation on VirusTotal.



While the sample is clearly malicious, my VirusTotal inspection does not suggest that the

sample belongs to any known family. No detections refer to Crypt0l0cker, TorrentLocker, a listed alias in the original Talos blog, nor <u>Teerac</u>.

Emsisoft	Trojan. GenericKD. 4899437 (B)	4.0.0.834	20170425
Endgame	malicious (high confidence)	0.4.1	20170419
ESET-NOD32	a variant of Win32/Injector.DNXZ	15308	20170425
F-Secure	Trojan. GenericKD. 4899437	11.0.19100.45	20170425
Fortinet	W32/Injector.DNXP!tr	5.4.233.0	20170425
GData	Trojan.GenericKD.4899437	A:25.12050B:25.9393	20170425
Kaspersky	HEUR: Trojan. Win32. Generic	15.0.1.13	20170425
McAfee	Artemis!BA6787041629	6.0.6.653	20170425
McAfee-GW-Edition	BehavesLike.Win32.Ransom.fc	v2015	20170425

With a file sample in hand, and no static indication that I have located Crypt0l0cker, I move onto FIRST to discover how similar it is to known files.

Exploring the Sample

Portable execut MS-DOS execut Binary file	table for 80386 (PE) table (EXE) [dos.ldw	[pe.ldw] /]		
Processor type				
MetaPC (disasse	mble all opcodes) (me	tapc]	•	Set
oading segment 0x00000000		Analysis	Kernel options 1	Kernel options 2
oading offset	0x00000000	 Indicator enabled 	Processor options	
Options				
Loading opt	ions	 Load resource 	rces	
Fill segment	gaps	 Rename DL 	L entries	
Create segr	nents	 Manual load 	d	
Create FLAT	l group	 Create implication 	orts segment	
Load as cod	e segment			

As the FIRST client code is an IDA Pro plugin, my first step was opening the file in my local IDA copy and allow auto-analysis. Upon completion, the start function was displayed in front of me at the graph view. I opened up the graph view context menu and requested FIRST lookups for all of the discovered functions.



After a minute, the FIRST display shows that 13 of the functions have been previously identified and uploaded.

Check All Function uery FIRST's server for function m f a function within this IDB matches vailable for you to select below to a	S a signature apply to you	found in	n FIRST then it elect the funct	and its metadata ion you wish to a	a will be pply existing		App	blied
etadata to in order to view the pos	Matches	es. Rank	Similarity	Prototype		i	Engines	ected ^
> 0x0040360e - sub 40360E	1						g	
> 0x00405a5f - sub_405A5F	1							
0x00405b78 - sub_405B78	1							
0x00405f15 - sub_405F15	1							
0x00405ad6 - sub_405AD6	1							
> 0x00405838 - sub_405838	1							
> 0x00405de3 - sub_405DE3	1							
> 0x0040564f - sub_40564F	1							
> 0x00405633 - sub_405633	1							~
<							>	•
latched 13 out of 83 functions					Show of	only	"sub_" funct	ions
					Select	High	est Ranked	
inished searching FIRST					Apply		Cancel	

Expanding the matched functions displays the metadata associated with that function, including a proposed name and function prototype. Notice that the files detected in this installer have been named to draw attention to the fact that these functions are known to be

in NullSoft Installers. I had previously marked up a different NullSoft Installer before and uploaded significant functions from it to assure that I would not do so again. In general, a malware analyst is wasting any time spent inspecting such a file. Identifying when a packer is in use and moving along to the true payload is a much better use of time.

f a function within this IDB matches vailable for you to select below to a netadata to in order to view the pos	a signature apply to you ssible match	found ir r IDB. S es.	n FIRST then it elect the funct	t and its metadata will be tion you wish to apply existing		Applie Selec	ed te
Function	Matches	Rank	Similarity	Prototype	i	Engines	^
Ox0040360e - sub_40360E	1						
> NullSoft_UnpackPayloa		6	75.0%	intcdecl()		Mnemonic	
 0x00405a5f - sub_405A5F 	1						
> NullSoft_ReadRegValue		7	90.0%	intstdcall(HKEY hKey,		BasicMaskir	
✓ 0x00405b78 - sub_405B78	1						
> NullSoft_BoundedString		7	90.0%	intstdcall(LPSTR lpStri		ExactMatch	
> 0x00405f15 - sub_405F15	1						
> 0x00405ad6 - sub_405AD6	1						
> 0x00405838 - sub_405838	1						~
<						>	
latched 13 out of 83 functions				Show	only tHigł	"sub_" functio nest Ranked	n

Check the Select Highest Ranked checkbox and click Apply. The function names get applied across the database and we can see clearly that the sandbox analysis was correct. This file is a packer and we need to extract the original.

f Functions window	8	×	F Functions window	8	×
Function name		^	Function name		^
f sub_4029E0			f sub_4029E0		
<u>f</u> sub_4029FD			f sub_4029FD		
f sub_402A3D			f sub_402A3D		
f sub_402AF2			<u>f</u> sub_402AF2		
<u>f</u> sub_402B07			<u>f</u> sub_402B07		
<u>f</u> DialogFunc			<u>f</u> DialogFunc		
<u>f</u> sub_402BC5			f sub_402BC5		
f NullSoft_IntegrityCheck			<u>f</u> sub_402C29		
<u>f</u> sub_402E62			<u>f</u> sub_402E62		
<u>f</u> sub_403055			<u>f</u> sub_403055		
<u>f</u> sub_40306B			<u>f</u> sub_40306B		
<u>f</u> NullSoft_CreateTempFile			<u>f</u> sub_403082		
<u>f</u> start			<u>f</u> start		
<u>f</u> sub_403534			<u>f</u> sub_403534		
<u>f</u> sub_40355E			<u>f</u> sub_40355E		
<u>f</u> sub_403579			<u>f</u> sub_403579		
<u>f</u> sub_4035AE			<u>f</u> sub_4035AE		
<u>f</u> sub_4035CC			<u>f</u> sub_4035CC		
<u>f</u> NullSoft_UnpackPayloadAndRun			<u>f</u> sub_40360E		
<u>f</u> sub_4038D3			<u>f</u> sub_4038D3		
100010					

Unpacking the Sample

I admit that at this point I cheated to perform the unpack. From previous extraction of Crypt0I0cker files protected with NullSoft I already knew that the install script consisted of consuming multiple encrypted blobs, internally decrypt the payload, and run it via <u>Process</u> <u>Hollowing</u>. As such, allowing it to run debugged and breaking on WriteProcessMemory should present the payload buffer to me.

There was a complication though, because the install script loaded and unloaded System.dll many times. The ModLoad notification caused the debugger to consume the majority of the process cycles, effectively causing a denial of service against the debugger. I allowed this system to run for over an hour without getting beyond this delay.

Command Importance HodLoad: /badUUUU /bDebUUU C:\Vindovs\SysUVb64\GDI32.dl1 ^ HodLoad: 76810000 7690d000 C:\Vindovs\SysUVb64\GDI32.dl1 ^ HodLoad: 76510000 7669e000 C:\Vindovs\SysUVb64\SHELL32.dl1 ^ HodLoad: 76510000 7669e000 C:\Vindovs\SysUVb64\SHELL32.dl1 ^ HodLoad: 76510000 742a7000 C:\Vindovs\SysUVb64\SHELL32.dl1 ^ HodLoad: 75cf0000 75e09000 C:\Vindovs\SysUVb64\School.cl32.dl1 ^ HodLoad: 75cf0000 75e09000 C:\Vindovs\SysUVb64\School.cl32.dl1 ^ HodLoad: 76a10000 76ac1000 C:\Vindovs\SysUVb64\School.dl1 ^ HodLoad: 74a0000 76ac1000 C:\Vindovs\SysUVb64\School.dl1 ^ HodLoad: 74a0000 76ac1000 C:\Vindovs\SysUVb64\School.dl1 ^ HodLoad: 74a0000 76ac1000 C:\Vindovs\SysUVb64\School.dl1 ^ HodLoad: 76ac0000 76ef4000 C:\Vindovs\SysUVb64\School.dl1 ^ HodLoad: 7490000 74962000 C:\Vindovs\SysUVb64\School.dl1 ^ HodLoad: 7490000 74962000 C:\Vindovs\SysUVb64\School.dl1 ^ HodLoad: 7490000 74962000 C:\Vindovs\SysUVb64\School.cl1 ^ HodLoad: 7490000 74962000 C:\Vindovs\SysUVb64\School.cl1 ^ HodLoad: 74900000 74962000 C:\Vindovs\SysUVb64\School.cl1 ^ HodLoad: 74900000 74962000 C:\Vindovs\SysUVb64\School.cl1 ^	E winit.exe E winiogon.exe E winiogon.exe E dum.exe E winiogon.exe E winiogo.exe D procesp.exe D procesp.64.exe D procesp.64.exe D procesp.64.exe D procesp.64.exe D procesp.64.exe	0.28 0.06 0.19 0.82 0.03
Ln 0, Col 0 Sys 0: <local> Proc 000:a70 Thrd 000:9e0 ASM OVR CAPS NUM</local>	CPU Usage: 1.75% Comm	it Charge: 11

By disabling the ModLoad notification via `<u>sxi Id</u>` I could get my debugger to allow the System.dll file to be loaded without triggering the significant extra processing. Crypt0l0cker then was able to spike up to 99% of the CPU use to, rather than the debugger holding on to 80%.



I dumped out the PE image file and prepared to continue with FIRST.

Exploring the Real File

Portable execut MS-DOS execut Binary file	table for 80386 (PI table (EXE) [dos.lo	E) [pe.ldw] lw]		
Processor type MetaPC (disasser	mble all opcodes) [n	netapc]	•	Set
Loading segment	0x0000000	Analysis	Kernel options 1	Kernel options 2
Loading offset	0x00000000	✓ Indicator enabled	Processo	r options
Options	ions	✓ Load reso	ources	
 Fill segment Create segn 	gaps nents	Rename IManual lo	DLL entries ad	
Create FLAT	l group	Create im	ports segment	

Again, the first step of using FIRST was opening the extracted file in IDA Pro. This file was built as a Windows GUI file on top of the Visual Studio C runtime. Thus, the runtime was identified during auto-analysis and I was left with a graph view displaying the _WinMain@16 function. Using the FIRST command from the context menu, I checked for the existence of metadata for just that one function. It was discovered as Crytp0l0cker_WinMain@16. Looking pretty likely that this is, in fact, Crytp0l0cker.



With confidence that FIRST will be useful, since it had a result for _WinMain@16, I began the search for the full file. At 436 functions this is not a quick lookup, so go get a fresh cup of coffee and let FIRST work. Since this file uses a known runtime, the runtime files are also known to FIRST. You can filter those functions with the Show only "sub_" functions checkbox.



After FIRST completes 78 new function markups are applied out of 295 total known functions. With 78 Crytp0l0cker_* functions identified, you can now dig in on the shoulder of days of professional malware analysis.

Conclusion

FIRST provides the ability to share your work from one file to a similar file, whether that other file is a previous or future version or even an additional step in the module execution. When opening up a new file, FIRST can provide hints as to whether the file is interesting or just needs to be unpacked. When finally extracting a new, embedded binary, FIRST can migrate your notes from the current file to the shared code in the new file. Use FIRST to save your notes, share your discoveries, and speed up your next analysis.

IOC

File Hash

d845e4f2292ba78a993dbbf6f1317894ce1a795c096d7959f3d718e583f1cea3