Analyzing a Brute Ratel Badger

blog.spookysec.net/analyzing-brc4-badgers/

09 Jul 2022

Now a days <u>Brute Ratel</u> (sometimes called the "Angry Monkey C2") seems to be a hot topic within the information security community. There's been lots of drama surrounding the author (<u>ParanoidNinja</u>), rumors of the C2 being backdoored, and even some <u>blog posts</u> from well known and respected individuals within the security community indicating that the C2 framework is potentially being used by APT29 (aka the Russian State Sponsored groups).

So, with all these controversies, where do we go from here? Well, validating the claim that the C2 Framework is backdoored can be quite difficult to prove as that would involve me spending several thousand dollars to acquire the framework itself... So, that's not exactly feasable. I can however get the next best thing. A Brute Ratel Beacon, or Agent (or as they like to call it, a "Badger").

Acquiring a Badger for Analysis

How can we do this exactly? Fortunately, I have a VirusTotal Enterprise license! This means we can pull down (download) a publicly tagged "Brute Ratel" sample from the community. To do so, we're going to use a search for something like **Comment:"Brute Ratel"** and see if we get any hits...

Σ	comment:"Brute Rate!"									ŧ	⊊ Help	Q	<u></u>
Q			≓ FILES 6/6 ▲ 90 c	days									
¢							~	Sort by -	Export -	Tools •	Help -		
×°		3	1ACF370188A8B9AFBCF6A4EC5029C3E19C9476 ③ ③ ④ No meaningful names peexe assembly overlay signed 64	641A2D9CE3CE56D71C1F576C869		32 / 60	284.92 KB	2022-06-26 14:14:28	2022-06-26 14:14:28	1	s ext		
(=)			58AE9193802E98AF17E6859E3FD8E3E9319C	5027726D60802E3E82D30D14D46	2.44 MB	2022-06-15 15:16:20	2022-06-15 15:16:20	1					
9		3	AD53495851BAFC48CAF6D2227A434CA2E0BEI © badger_x64.exe peexe overlay runtime-modules check	250.79 KB	2022-05-20 20:45:39	2022-05-20 20:45:39	1	SL EXE					
			A2876E9175410866F6719F80EE44B95539607 (a) (volumes/17_05_2022/version ped1) (04bits (assembly)	253.50 KB	2022-05-19 13:06:58	2022-05-19 13:06:58	1						
Þ			FC7B8E1854D54CE8F1DE8CC95976881C7A85 Sologe Roshan_CV.iso isoimage contains-pe	C7926C03172A3DDAA672690042C		26 / 58	4.62 MB	2022-05-19 13:06:36	2022-05-19 13:06:36	1	100		
			50103C1AEC2F2EF06E700879968C45DF47449	9348D66266A6E8802D70E35236E		18 / 57	270.57 KB	2022-05-19 10:20:36	2022-05-19 10:20:36	1			
ă													
			VirusTotal Contact Us Get Support How It Works ToS Privacy Policy Blog Releases	Community Join Community Vole and Comment Contributors Top Users Community Buzz	Tools API Scripts YARA Desktop Apps Browser Extensions Mobile App	Premiur Intelliger Hunting Graph API v3 1	m Services nce v2		Documentation Intelligence Hunting Graph API v3 v2 Use Cases	1			

Suprise Suprise, we got six hits! Let's go with the most obvious one, <u>badger_x64.exe</u> (SHA256 Sum:

3ad53495851bafc48caf6d2227a434ca2e0bef9ab3bd40abfe4ea8f318d37bbe).

Lab Setup

For this lab, we will be using REMWorkstation + REMnux. Here's a diagram that breaks down the lab setup:



- REMWorkstation has the IP Address of 192.168.128.12
- REMNux has the IP Address of 192.168.128.10
- Default Gateway has the IP Address of 192.168.128.2
- REMNUx can route to 192.168.128.2, but the route is not configured.
- If REMNux is configured to route to the Default Gateway, outbound traffic to the internet is allowed

In addition:

- REMNux will have an iptables rule that will accept all and any traffic going into it.
- REMNux will be running FakeDNS and iNetSim
- REMNux will be running WireShark
- REMWorkstation will be running Fiddler

And thats our lab!

Dynamic Analysis - Malware Detonation

Now that we have our sample acquired, and you're familiar with my lab setup, let's double click some EXEs!

Progress Telerik Fiddler Web Debugg	per						- 🗆 ×			Reces	ss Monitor - Sysint	emals: www.sysinternals.com				- 0 ×
File Edit Rules Tools View Help										File Edit	t Event Filter	Tools Options Help				
# Result Protocol	Host URL			Body Cathing	∮ AutoResponder	Composer	Fidder Orchestra Beta				- R 🖻	7708				
10 200 HTTP	Tunnel to 159.65.1	86.50:443		0	👸 FidderScript	🗄 Log 🔲	Filters 🔚 Timeline									
10 22 200 HTTP 1	Tunnel to 159.65.1	26.50:443		0	Get Started	🖉 Statutos	inspectors 🙀			TIM P	TOCESS N	PID Operation	Path	Result	Detail	TID Image Par
10 25 200 HTTP	Turnel to 159.65.1	26.50:443			Headers Textiler	w SyntaxVev	WebForms			3:10 🐂	Explorer	496 BRegQueryKey	HKCU/Software/Classes	SUCCESS	Query: Name	2352 C:WINDO
E 20 200 HTTP	Turnel to 159,65.1	26.50.943			Herview Auth	Cookies Raw	JSON XML			3.10 🐂	Explorer	496 ERegQueryKey	HKCU/Software/Classes	SUCCESS	Query: HandleTags, Han	2352 C.WINDO
ff 34 200 HTTP	Turnel to 159,65.1	26.50.443			Request Headers					3:10 🠂	Explorer	496	HKCU/Software/Classes	SUCCESS	Query: HandleTags, Han	2352 C:WINDO
1 37 200 HTTP 1	Tunnel to 159.65.1	86.50.443		0	CONNECT 159.65.18	86.50.443 HTTP/1.0				3:10 🠂	Explorer	496 BRegOpenKey	HKCU/Software/Classes/CLSID/(56AD4C5D-B908-4F85-8FF1-7940C29B3BC	NAME NOT	Desired Access: Read	2352 C:/WINDO
@ 40 200 HTTP 1	Tunnel to 159.65.1	86.50.443		•	Cache		^			3:10 🠂	Explorer	496 BRegOpenKey	HKCR\CLSID\[56AD4C5D-B908-4F85-8FF1-7940C29B3BCF]\Instance	NAME NOT	Desired Access: Read	2352 C:/WINDO
M-13 200 HTTP	Tunnel to 159.65.1	86.50:443			Cleat					3.10 🦬	Explorer	496 BRegQueryKey	HKCU/Software/Classes	SUCCESS	Query: Name	2352 C./WINDO
6 40 200 HTTP	Turnel to 159.65.1	00.00.00			User-Agent: Moz	alla/5.0 (Mindevis NI	20.0; Wri64; x64) 4;		3+45149505	3:10 🍗	Explorer	496 #RegQuervKey	HKCU/Software/Classes	SUCCESS	Query: HandleTags. Han	2352 C:/WINDO
₫ \$2 200 HTTP	Tunnel to 159.65.1	86.50:443			Entity					3 10	Explorer	496 BRegQuervKey	HKCU/Software/Classes	SUCCESS	Query HandleTags, Han	2352 CIWINDO
商 SS 200 HTTP 1	Tunnel to 159.65.1	86.50:443		0	Tremont	10				3 10	Explorer	496 BRegOpenKey	HKCU/Software/Classes/Applications/Procmon64 exe	NAME NOT	Desired Access: Read	2352 C1WINDO
65 50 200 HTTP	Tunnel to 159.65.1	86.50:443			Connection: Kee	p-Aive				3 10 🍗	Explorer	496 BRegOpenKey	HKCR\Applications\Procmon64 exe	NAME NOT	Desired Access: Read	2352 C/WINDO
m 61 200 HTTP	Tunnel to 159.65.1	26.50:443			Host: 159.65.15	55.50:443	*	- n ×		3.10	Emlorer	196 MR DegOpenitor	HKCI ISoftware/Classee	SUCCESS	Quary Name	2352 C WINDO
ED 04 200 1110					<		,			2.10	Explorer	406 BRDadOuankau	HKCI I Software/Classes	SUCCESS	Query HandleTage Han	2352 C-WINDO
					Transformer H68	ders Text/iew	Syntaxilew	rear (Colum)		2.10	Explorer	406 BRogQuarynay	HKCI ISoftware/Classes	GINDERG	Query Name	2252 C WINDO
¢					> InageView HexN	levi Webilev	Auth Cadling	and the second s		0.10	Capitoriei	too million Construction	LIKOLIO de la constructione de la constructione de la construction de	NAME NOT	Query, Name	2002 0.000000
QuidExed ALT+Q > type HELP to learn mon	10				Cookies Raw	250N XHL				3.10	Explorer	490 ERegopenkey	HKCO/software/classes/applications/Procinionov.exe	NAME NOT	Desired Access: Read	2352 C.WINDO
Filters Hide Tiddler2.com' Hide 'ctidi.win	ndowsupdate.com					1.11				3:10	Explorer	490 E Regupenkey	HKCHWppications/Hocmonio4.exe	NAME NUT	Desired Access: Read	2352 CHWINDO
Capturing V Al Processes	1/35 The	is a HTTPS Decrypting tur	nel to '159.65.186.	50:443'.						3:10	Explorer	496 EACreater-se	Chusersi KEMWAppUata/Local/Temp/Procmono4.exe	SUCCESS	Desred Access: Read A	2352 CHWINDO
senss.exe	292 ASLR	System		368 kB NT	AUTHOR/TY/SYSTEM	Wi	ndows Session Manager			3.10 🐂	Explorer	496 EQueryBasicInforma	C1UsersIREMIAppData/Local(Temp/Procmon64.exe	SUCCESS	CreationTime: 7/9/2022	2352 C.WINDO
Memory Compression	1464	System		216 kB NT	AUTHOR/TY/SYSTEM					3:10 🦷	Explorer	496 LocioseFile	C:IUsersIREM/AppData/Local/Temp/Procmon64.exe	SUCCESS		2352 C:/WINDO
Interrupts			1.61	0		Int	emupts and DPCs			3:10 🗾	ctimon exe 4	 BRegQueryKey 	HKCU	SUCCESS	Query: HandleTags, Han	920 C.WINDO
C1753.4509	392 ASLR	System		1.54 M8 NT	AUTHOR/TY/SYSTEM	Cia	int Server Runtime Proce	55		3:10 📝	ctfmon.exe	 BRegOpenKey 	HKCU/Software/Microsoft/InputPersonalization/TrainedDataStore	NAME NOT	Desired Access: Notify	920 C:WINDO
W wrinitese	468 ASLR	System		1.21 M8 NT	AUTHORITY/SYSTEM	We	ndows Start-Up Applicati	in .		3.10 📝	ctimon.exe	11 IIIRegQueryKey	HKLM	SUCCESS	Query: HandleTags, Han	920 C.WINDO
> III: services.exe	544 ASLK	system	0.02	4.77 M8 N1	AUTHORITYSYSTEM	Set	vices and Controller app			3:10 📝	ctfmon.exe <	11 BRegOpenKey	HKLMISoftware/Microsoft/Input/Settings	SUCCESS	Desired Access: Read	920 C:/WINDO
Bass eve	572 ASLR	system	0.14 448 8/5	7.24 M8 N1	AUTHORITYSYSTEM	Los	a security Authority Pro	cess		3:10 📝	ctfmon.exe 4	1 BRegQueryKey	HKCU	SUCCESS	Query: HandleTags, Han	920 C:WINDO
T crist me	454 451 8	Suitem	0.23 3.66.48/9	1.57 MB 101	ALTHORITOSYSTEM	Ci	ont Server Runtime Proce			3:10 🖉	ctfmon.exe	1 BRegOpenKey	HKCU/Software/Microsoft/Input/Settings	NAME NOT	Desired Access: Read	920 C.WINDO
V Winlogon.exe	580 ASLR	System		2.84 M8 NT	AUTHORITY/SYSTEM	W	ndows Logon Application			3.10. 📝	ctimon exe	1 BRegQuervKev	HKLM/SOFTWARE/Microsoft/Input/Settings	SUCCESS	Query, HandleTags, Han	920 C.WINDO
Fontdryhost.exe	736 ASLR	Low		4.2 M8 For	t Driver Host/UMFD-1	Us	ermode Font Driver Host			3-10 📝	Ctfmon exe	1 BRegOpenKey	HKLMISOETWARE/Microsoft/Input/Settingsiproc 1/ioc 0409/im 1	SUCCESS	Desired Access: Query V	920 C WINDO
dwm.exe	992 ASLR	System	0.93	224.89 M8 Wr	ndow Manager\DWM-1	De	sktop Window Manager			3.10	ctimon erre	1 BRedQueryValue	HKLMISOFTWARE/Microsoft/Input/Settings)proc. 1/loc. 0409/im. 1/Expressivel	SUCCESS	Type: REG_DWORD_Le	920 C WINDO
Y n explorer.ese	496 ASLR	Medium	0.74	47.4 MB DES	SKTOP-2C3IQHO/REM	W	ndows Explorer			3.10	ctimon eve	1 BRenCloseKey	HKLMISOFTWARE/Microsoft/Input/Settings/proc. 1/loc. 0409/im. 1	SUCCESS	.,,	920 C WINDO
sea biloctrey 🔛	6760 ASLR	Medium	0.09 1.54 kB/s	31.92 MB DE5	SKTOP-2C3IQHO\REM	VM	Iware Tools Core Service			2.10	Cottmon ovo	1 BRDogClocoKov	HVI MSOETWADEIMerosoftilaput/Softings	SUCCESS		020 C WWINDO
Y Procmon.exe	2600 ASLR	Medium		4.94 MB DES	SKTOP-2C3IQHO/REM	Pro	cess Monitor	_		2.10	offeren erro	1. BRDogOutserkey	LIVEN SOLEWARE INCODUCINED CORRECTING	SUCCESS	Overa HandleTage Han	020 CUMINDO
Procinonii 4 exe	6452 ASLR	High	0.72 289.95 88/9	82.5 M8 DES	SKTOP-2C3IQHOVREM	Pro	cess Monitor	_		2.10	etfmon ene	1	Life Micodeaner/Manageditions diCollings	CLOCESS	Query, Handlerags, Harr.	020 C.IWINDO
sato s495851bafc48cafid	0,004	Medium	0.41 \$.06 kB/s	6.22 M8 DE1	skrop-zcarQH0/REM		alarse Text			3.10	Cumon exe	1. Brogopenkey	Inconsormal emicrosormipacaetings	CUCCESS	Overal UsediaTerra Use	920 C.WINDO
· La suberne_stor.ese	door ASLR	High		12.21 MB DE	SETOP-2CELOPIO (REM	su	arre lest			3.10. 2	rctimon.exe	T WritegQuerykey	HKCU	SUCCESS	Query: Handle Lags, Han	920 C.WVINDO
Fidder car	4236 ASLE	Medium	443 533184	51 87 MR DF	SKTOP-2CRIDHOVEEM	64	der			3.10 🜌	rctimon.exe +	II #rkegupenKey	HKCUISottware/Microsott/input/Settings	NAME NOT	Desired Access: Read	920 C.WINDO
Papersk-lacker me	3616 ASLR	Medium	1.15	20.48 MB DE!	SKTOP-2CRIDHO/REM	Pro	cess Backer	_		3:10 🜌	ctfmon.exe 4	11 #RegQueryKey	HKLMISOFTWARE/Microsoft/Input/Settings	SUCCESS	Query: HandleTags, Han	920 C:/WINDO
										3:10 🛃	ctfmon.exe 4	1	HKLMISOFTWARE/Microsoft/Input/Settings/proc_1/loc_0409/im_1	SUCCESS	Desired Access: Query V	920 C:WINDO v
CRITICION 11 1996 Diversion memory	1 91 GB (47 696)	Descenter 122								<						>
crocologic musical memory.	(#1.90%)	1000000-102						2		snowing 62	co, seu or 626,903 e	vents (997k) Backed by via	tual memory			

So, right off the bat, we can see some beacons to 156.65.186.50 over HTTPS. Looking at these requests in Fiddler, we can see that the sample is using the user agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.93 Safari/537.36 with no extra headers.

🔚 FiddlerScript	🗏 Log		Filters	🚍 Timeline						
Get Started 🛞 Statist	ics 🔍 Inspectors	🐐 AutoResp	onder 🛛 🗹 Compos	er FO Fiddler Orchestra Beta						
Headers TextView SyntaxV	iew WebForms HexView	w Auth Coo	cies Raw JSON	XML						
CONNECT 159.65.186.50:443 HTTP/1.0 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0 Host: 159.65.186.50:443 Content-Length: 0 Connection: Keep-Alive Pragma: no-cache										
A SSLv3-compatible Clier	tHello handshake was	found. Fiddler	extracted the para	meters below.						
Version: 3.3 (TLS/1.2) Random: 62 C9 D3 59 81 05 25 42 30 82 AA 23 56 96 33 87 BB 3B 22 C6 0B 17 37 6B 50 F5 A1 5F 00 D3 D1 AB "Time": 10/3/2017 1:31:14 PM SessionID: empty Extensions: status_request OCSP - Implicit Responder supported_groups x25519 [0x1d], secp256r1 [0x17], secp384r1 [0x18]										
Find (press Ctrl+Enter to highlig	iht all)			View in Notepad						
Find (press Ctrl+Enter to highlight all) View in Notepad Transformer Headers TextView SyntaxView ImageView HexView WebView Auth Caching Cookies Raw JSON XML JSON XML ImageView HexView WebView Auth Caching Cookies Raw JSON XML ImageView HTTP/1.0 200 Connection Established FiddlerGateway: Direct StartTime: 15:13:29.434 Connection: close Government Government <t< td=""></t<>										

This is suprisingly bare. Let's pivot over to iNetSim and see whats going on over there.



н.

On that side, we can see a little bit more. The file that the "Badger" requested is /admin, and there is also some POST data that we missed!

Let's see if we can find that in Fiddler... Unfortunately, I could not find the request in Fiddler, I'll have to revert and redetonate the sample in a bit...

Edit: Fiddler actually caused some issues w/ cutting the POST data off to inetsim :(.

Procmon/ProcDot Analysis

For now - Let's move over to ProcMon and ProcDot and see what the badger is looking for.



Starting out, this is an absolutely massive graph. Let's start from the top and work our way down.

At the top:



It appears that the badger is first checking to see if there are any registry keys correlated to a proxy on the system. Since no proxies are in place, BRC4 likely foudn nothing.

On the far right, we can see a couple of cached web page responses saved to disk. If you'd like to read that data - all it contains is the iNetSim HTTP Response.



Moving on down the graph, we can see another read attempt on another registry key relating to proxies:

HKCU\Software\Microsoft\Windows\CurrentVersion\InternetSettings\Connections\DefaultConnectionSettings

One interesting thing I'd like to point out is the Badger is leveraging a bunch of ThreadCreates and ThreadOpens to potentially confuse AV or EDR.

Zooming out, all the black diamons are all new threads and Thread ID Numbers.



Scrolling down a bit more, this pattern continues. More Threads being created to read registry keys relating to proxies:



Back to iNetSim

Now that we know a bit more about what the program is trying to do, let's go back to iNetSim and read the POST data from the Web Server.

All of the POST data is stored in <a>/var/lib/inetsim/postdata/*. I hope that helps someone in the future... :)



Let's bring the input into CyberChef and decode the Base64.

Recipe		Î	Input	start: 0 end: 522 length: 522	length: 524 lines: 1	+	• •	Î =
From Base64	(9 11	6MDF4WQYm89721dUqAclEShq9fp+1XoNOHY71VPrakNONZ4yIW0sBg3MedVz6Q7pT+eCYzxBMR/virWNIFPkS64SM63368gzda6Bgy6vphQdmqvGWqAurG3 YQfAR0gnSnL9Y27EiuvTEGUEgFH4AT364sK1Vb21/Uz57Zpoh4ypEo/eWILBDyMryA6pyyFKNQPS0g8eXvM4XnL/p32hksQoT13zVyf1QYQwioegnPnhH	+Smf8VrIzSu fNTTNGchnYF	Z+/g8StzgP7 erggQ3u	eZgcRFxR	ghSkdq3nWp	pZHYcyOhY
Alphabet $N - ZA - Mn - 2a - m\theta - 9+ /=$		*	/grMAeEzbfllAghKd+puf42KuYvIYeSDyAeBB0e703Gj9M4VUYxOONVORp7nIY9j1n//ZlGjtM/+g7xlbvlcnksUDx+STy5VXm /jPaoMKD595oZyXc26locEn8Y2lr1MHonZnOKhW300d1LHk9ZAeqoREHT0FFixPL0psv2+zu4qwzaEeN8Jde/vFG+xiPbScDoPPsL3XSpffwqBtd7ENPg	-				
Remove non-alphabet chars								
			Output	start: end: 3 length: 3	0 time: 3 31 length: 3 31 lines:	^{ns} ⁹¹	6 🖬	n 0
			e. a. EII(Øj.tÜrD[.6C>T A.8004.1%v=.;YÜ868L>Ü.EI.i.d08%88H ±.89g¥-EEF*¶veråeI8.%Öay=b.; <låf>pGk81³-¤D¤.Ü.45%h. .)8))H£ky.h1xEi&n.oüE\$y.BY^.b.T.586**&x*ix518A6Ü.1#*'[5.*8.08µ.3+];m.J.'=aJ.'=abx.6' \\$I*.Ü.«cI*,±P.Ü0;FE1%6A5x.</låf>	.P°e.āÿ.ĀDÒ ālG:iòýs	Ö.i.K3'o.!.	LtmH. Vë{.ú	"Ö.ÄltSùñ¥	¥æ"
			E9.51.Acb[2860w.0,1.0.]h£.^dw.JµaL]00YlDT.*WJ0.cG.o]c.tk.A ^{w.} 0.06/ .1A^.].É.E0 «:0		,	,		

Searching for Encryption in APIMonitor

Interesting! The POST Data is encrypted. I think I know a trick or two that could help us decode this. To do so, we'll need to hop into API Monitor and hook into the process and observe the API Calls the badger is performing. We're looking for a call to Microsoft's Cryptographic API *or* a call to the HTTP APIs as we know some cryptographic function performs before the POST data is sent...

	290875	4:04:54.926 PM	2	wininet.c	111	memcpy (0x000000002877a10, 0x0	0007ffcce1a7c60, 5)						
	290876	4:04:54.926 PM	2	wininet.c	III	memcpy (0x000000002877a15, 0x0	0007ffcce1a7c5c,3)						
	290877	4:04:54.926 PM	2	wininet.c	HI III	memcpy (0x000000002877a18, 0x0	000000026ab5c0, 13)						
	290878	4:04:54.926 PM	2	wininet.c	dii 👘	memcpy (0x000000002877a25, 0x0	0000000285ebc0, 8)						
	290879	4:04:54.926 PM	2	KERNELB	ASE.dll	RtIUTF8ToUnicodeN (NULL, 0, 0x00	8ToUnicodeN (NULL, 0, 0x0000000059be7b8, "https://159.65.186.50/admin", 30)						
	290880	4:04:54.926 PM	2	KERNELB	ASE.dll	RtIUTF8ToUnicodeN ("D", 60, 0x000	F8ToUnicodeN("□", 60, 0x0000000059be7b8, "https://159.65.186.50/admin", 30)						
	290881	4:04:54.926 PM	2	KERNELB	ASE.dll	memcpy (0x0000000059be49a, 0x0	cpy (0x0000000059be49a, 0x00007ffce2fd8708, 10)						
	290882	4:04:54.926 PM	2	KERNELB	ASE.dll	memcpy (0x000000002878b10, 0x0	cpy (0x000000002878b10, 0x0000000059be49a, 56)						
1	290883	4:04:54.926 PM	2	KERNELB	ASE.dll	RtIUnicodeToUTF8N (NULL, 0, 0x00	JnicodeToUTF8N (NULL, 0, 0x00000000059be830, "https://159.65.186.50/admin", 56)						
	290884	4:04:54.926 PM	2	KERNELB	ASE.dll	RtIUnicodeToUTF8N (**, 28, 0x0000	0000059be830, "https://159.65.186	.50/admin	, 56)				
												Hex B	uffer: 60 by
		Pre-Call Value			Post-Call Val	ue		E					a 😰 🖡
tin	ation	0x00000000287	8140 "0"		0x00000000	2878140 "https://159.65.186.50/adm	in"					0000	68 00
By	teCount	60			60							002a	2f 00
ıal	ByteCount	0x000000059b	e7b8 = 0		0x00000000	59be7b8 = 60							
		0x0000000287	7a10 "https	://159.6	0x00000000	2877a10 "https://159.65.186.50/adm	in"						
Jn	t	30			30								
-								Find				×	
					STATUS_SUC	CESS							
								Find what:	RtIUTF8ToUnico	deN	Eind Ne	xt	
										Direction	Cance	1	it
t		Location									Ganoc		Loadin
86		MultiByteToWide	Char + 0x2	25e				Match case				ories:	
d	d9 InternetCreateUrIW + 0x3d89											La	ables:
20		144-0										DLLs	

By searching for a common Windows API (RtIUTF8ToUnicodeN), we can quickly find where some data conversion is taking place to give us a good starting point of reference.

-	NTSTATUS	🖗 Return			STATUS_SUCCESS
Cal	I Stack: RtIUTF8ToUn	nicodeN (Ntdll.dll)			
#	Module	Address	Offset	Location	
1	KERNELBASE.dll	0x00007ffce2e7	0x4d18e	MultiByteToWideChar + 0x25e	
2	wininet.dll	0x00007ffccdff	0x213d9	InternetCreateUrIW + 0x3d89	
3	wininet.dll	0x00007ffccdff	0x22826	HttpOpenRequestW + 0x12a6	
4	wininet.dll	0x00007ffccdff	0x21953	HttpOpenRequestW + 0x3d3	

Looking at the CallStack, we see some lovely Windows API calls that look very close to what we need. Since some sort of technique is being used to dynamically resolved the APIs needed is being used, let's back off of APIMonitor and move over to a Debugger.

Pivoting to x64Dbg

I have setup x64Dbg to use counter-antidebugging techniques using ScyllaHide, so if there are any techniques implemented, we won't have to worry about them.

After letting the program run for a while, I set a breakpoint on a couple of the common HTTP APIs. We got a hit on InternetOpenW; in my suprise, in the stack window, here we are. We have the unencrypted data starting at us!



It appears to be some JSON that looks like so:

```
"desktop-2c3Iqh0",
    "wver":"x64/10.0",
    "arch:"x64",
    "bld":"16322",
    "p_name":"<base64 blob>",
    "uid":"REM",
    "pid":""
}
```

The Base64 glob is still relatively interesting to me, p_name, could this mean program_name? Let's decode it!

	Las	st build	d: A day ago	Option	s 🌣	Abo	ut / S	upport	0
Recipe	2 🖬	Î	Input	length: 240 lines: 1	+		∋	Î	=
From Base64	0	П	QwA6AFwAVQBzAGUAcgBzAFwAUgBFAE0AXABEAGUAcwBrAHQAbwBwAFwAMwBhAGQANQAzADQAOQA1ADgA 3AGEANAAzADQAYwBhADIAZQAwAGIAZQBmADKAYQB1ADMAYgBKADQAMABhAGIAZgB1ADQAZQBhADgAZgA	NQAxAGIAYQBmAGMANAA4 AZADEAOABkADMANwBiAGJ	AGMAY AZQAL	QBmAI	OYAZA ⊵AB1A	AyADI A==	AMgA
Alphabet A-Za-z0-9+/=		•							
Remove non-alphabet chars	Strict mode								
Remove null bytes	0	п							
			Output	time: 2ms length: 89	B	Ē	(f)	5	0
			• C:\Users\REM\Desktop\3ad53495851bafc48caf6d2227a434ca2e0bef9ab3bd40abfe4ea8f318c	lines: 1 137bbe.exe					-

It appears so! I set a BreakPoint earlier in the stack and let the execution flow to see if I could extract any more information from the Badger, doing so did yeild some extra results!



We have an auth token now and a more complete JSON blob.

```
{
"cds": {
    "auth":"2K4TBS7L9GK2C205"
    },
"mtdt": {
    "h_name":"DESKTOP-2C3IQH0",
    "wver":"x64/10.0",
    "arch":"x64",
    "bld":"16322",
    "p_name":"<base64 blob>",
    "uid":"REM",
    "pid":""
    }
}
```

Unfortunately, our analysis stops here as we don't have a live C2 server to observe interactions with. Though, we could explore *how* the badger interacts with the C2 server if we carefully observe how the badger parses the response from the C2 server. There is definately some hardcoded commands that we would be able to use to manipulate the badger itself with iNetSim.

I would have liked to have caught the Windows API that actually encodes/encrypts this data, so I could write a small decoder for the information if you have the badger; but it appears that wasn't meant for tonight :(

Basic Static Analysis

So, this section is going to be much shorter than the last, as I've already found the interesting C2 related data; Now, we're going to play an interesting game of "How good is Brute Ratel's Obfuscation Techniques"! The answer isn't very good.

To start, we're going to chuck the EXE into Cyberchef and look at some of the clear text ASCII values.

Recipe	8 🖬 🖬	Input	Name: 3ad53495851bafc48caf6d22 × 256,810 + 🗅 🔁 🔋 🔳
Remove null bytes	⊘ 11		Size: 256,810 bytes
-			Type: application/x-msdownload
		Output	start: 64 time: 41ms end: 2244 length: 211251 🖬 🗖 😭 🖛 🚼 length: 2180 lines: 813
		<pre>MZyy,@².⁵ !!Lf!This program cannot be \$PEd60.ba.55[*]#à@i /logiPH.OEGRDTSSPH.5]30FUIDH.L9GK2C20PH.[2KAT like PH.36 (KHTMPH.Kit/S37.PH.AppleWeDPH.4;xA (WindOPH.zilla/S.PH.0]443]MOPH.05.186.5PH.0]1] .P.P.P.P.P.P.P.P.P.P.P.P.P.P.P.P.P.P.P</pre>	<pre>run in DOS mode. text.8a P`.idata.5â.@@A. PH,n,/adminPH,R1 BS7PH,i/S37.36PH.93 SafarPH0.4430.PH.hrome/90PH.Gecko) CPH.L, 4) PH.0; WinGPH.ws NT 10PH.0 159.Pht P.P.P.P.P.P.P.P.P.P.P.P.P.P.P.P.P.P.P.</pre>
STEP 📃 🗾 BAKE!	Auto Bake	PH, IFreek.PH, ê.LocaPH, calAllocPH, onå.LoPH, calS locPH, i.HeapPH, HeapFreePH, Alloce.PH,HeapPH, i DeocoPH, pocci, DH, theochiddu, obs. CoPH, ulayand D	ectiPH, aveCritiPH, ion@.LePH, icalSectPH, lizeCritPH, .InitiaPH,ReAl ckCountPH,eGetTPH,SFileTimPH,temTimeAPH,GetSySPH,sSHeapPH,Get

HTTP Request Information

So, right off the bat, it's not looking so good. We can see a **lot** of interesting strings; we can see a lot of the HTTP POST information broken up into various strings. For example:

- /logi
- AppleWeb
- Kit/537
- 65.186.5
- 159
- 443

Some of these strings are incredibly meaningful! For example, putting together the bits 159.65.186.50 gives away our command and control server, and 443 gives away the port! How interesting...

Windows APIs

Looking a little bit lower, we can see some of the Windows APIs the program uses as well. They appear to be jumbled up, but still readable to the human eye. ,strtokPH,rncmp>.PH,en6.stPH,3.strlPH,..strcpyPH,rcat_SPH,and*.stPH,ntf&.srPH,#.spriPH,..signalPH,reallocPH,ran d..PH,mcmp..PH,_sÿ.mePH,mbstowcsPH,llocü.PH,aceø.maPH,ß.isspPH,Ê.fwritePH,¾.freePH,callocPH,atol..PH,ctime..PH, ort..asPH,mp..abPH,

._wcsicPH,wprintfPH,ï._vsnPH,snprintfPH,ocké._vPH,Ë._unlPH,Â._ultoaPH,_time64PH,_lock¶.PH,me64..PH,_localtiPH,t term..PH,.._iniPH,¾_errnoPH,time64PH,it§_cPH,_amsg_exPH,funcyPH,T__iob_PH,trlenWPH,yWO.lsPH,I.lstrcpPH,lQueryPH ,Ö.VirtuaPH,rotectPH,VirtualPPH,lterÔ.PH,eptionFiPH,ndledExcPH,e^a.UnhaPH,sGetValuPH,ead¥.TlPH,inateThrPH,..Term PH,eProcessPH,TerminatPH,Sleep..PH,ilter..PH,ceptionFPH,andledExPH,r.SetUnhPH,UnwindPH,lVirtualPH,ryÕ.RtPH,ctio nEntPH,ookupFunPH,tî.RtlLPH,reContexPH,RtlCaptuPH,TableÇ.PH,FunctionPH,&.RtlAddPH,CounterPH,formancePH,QueryPer PH,lFreek.PH,ê.LocaPH,calAllocPH,onå.LoPH,calSectiPH,aveCritiPH,ionØ.LePH,icalSectPH,lizeCritPH,|.InitiaPH,ReAl locPH,i.HeapPH,HeapFreePH,Alloce.PH,_.HeapPH,ickCountPH,e..GetTPH,SFileTimPH,temTimeAPH,..GetSysPH,ssHeapPH,Get ProcePH,ressì.PH,tProcAddPH,eW&.GePH,uleHandlPH,..GetModPH,tErrorPH,v.GetLasPH,readIdPH,urrentThPH,d-.GetCPH,Pr occssIPH,tCurrentPH,ess).GePH,rentProcPH,

(.GetCurPH,LibraryPH,e».FreePH,eeConsolPH,on,.FrPH,calSectiPH,terCritiPH,ion?.EnPH,icalSectPH,leteCritPH,..DeP, PH,.I.PH,.I.PH,0H.PH,êH.PH,àH.PH,ÌH.PH,ÌH.PH,ÂH.PH,H.PH,

¬Н.РН ¤Н.РН .Н.РН .Н.РН .Н.РН ~Н.РН tH.PH fH.PH \H.PH RH.PH HH.PH @H.PH 6H.PH .H.PH \$H.PH .H.PH .H.PH .G.

- VirtualProtect
- GetLastError
- GetModuleHandleW
- GetProcAddress

The more you keep looking, the more you see the pattern.

HTTP POST Data

Interestingly enough, you can actually find a lot of the HTTP POST Data that we had to work oh so hard to reverse engineer to find...

```
п,ремпп,югорпт, с этп,лазопт, п
РН,----РН,----РН,----РН,----РН,----РН,----РН,----РН,----РН,----РН,----РН,----РН,----РН,----РН,----РН,----РН,----
PH,----PH,+--PH, %S
PH, osedPH, ] C1PH,
[-PH, ST%sPH, SPOPH, %s%ls%PH, %ls%s%lsP, P, BFPH, 3#M?:XyMPH, bYXJm/PH, d%lsPH, %ls%PH, s%lsPH, ls%lPH,
%ls%PH,s%lsPH,ls%lPH,%ls%PH,s%lsPH,ls%lPH,%ls%PH,ls,PH,%s%PH,POSTPH,s%lsPH,ls%lPH,%ls%PH,ls%SPH,ls%PH,
%ls%PH,ls%SPH,%S%P,P,STPH,%lsPOPH,%lsPH,s%luPH,%S%lPH,S%lsPH,%ls%PH,ls%SPH,V2%PH,\CIMPH,ROOTPH,}
PH.":PH."{"PH.ze":PH.dfsiPH.","PH.e":"PH.fnamPH.","dPH.":"PH.hkinPH.:{"cPH."dt"PH."},PH.}}
PH.:""PH.pid"PH.","PH.d":"PH.,"uiPH.:""PH.ame"PH."p_nPH."",PH.er":PH.,"wvPH.:""PH.ame"PH."h_nPH.t":
{PH,"mtdPH,"},PH,h":"PH,"autPH,s":{PH,{"cdPH,":"PH,"bldPH,64",PH,":"xPH,archPH,0","PH_%SPH,%S
P,PH,Y@PH,SÈBPH,ls%PH,d]
%PH,E: %PH,ed [PH,failPH,oad PH,ownlPH,-] DPH,ls[PH,te
%PH,mplePH,d coPH,nloaPH, DowPH,[+]PH,%%)
PH..2f PH, (%0PH.s %SPH.tatuPH.ad SPH.wnloPH.] DoPH.s[+PH.d
%1PH,E: %PH,[-] PH,-+
DU
```

- arch
- bld
- fname
- h_name

Continuing our search, we may be able to learn more about the badgers capabilities. Looking at the screenshot above, towards the bottom, we can make out "Download Failed". Perhaps this badger has the ability to upload files to the server? Let's keep digging.

PH, ----PH, +--PH, %lsPH,: %SPH,adedPH,wnloPH,t doPH,nshoPH,creePH,+] SPH,eb[PH,05e7PH,b351PH,d-5dPH,-9cdPH,452dPH,a4a-PH,b5-FPH, 5be4PH, g1dPH, e/pnPH, imagPH, 2d.pngPH, 02d%2d%0PH, %d_%02d%PH, %02d%02dP, P, PH, %ls PH.ng: PH.erviPH.] QuPH.S[*PH.://%PH.LDAPPH.E. PH,otDSPH,//roPH,DAP:PH,g: LPH,ndinPH,r biPH,ErroPH,[-] PH,%lsPH,P://PH,LDAPH,textPH,gConPH,aminPH,ultNPH,defaPH,DSEPH,rootPH,P://PH,LDAPH,gue PH,taloPH,1 CaPH,lobaPH,g: GPH,ndinPH,r biPH,ErroPH,[-] PH,ap PH.g ldPH.ndinPH.r biPH.ErroPH.[-] PH.GC:PH.ue PH,alogPH, CatPH,obalPH,: GlPH,yingPH,QuerPH,[*] PH,:XyMBFPH,XJm/3#M?PH,lsbYPH,+ %РН,----РН,----РН,----РН,----РН,----РН,----РН,----РН,----РН,----РН,----РН,----РН,----РН,----РН,----РН,----РН,----PH,----PH,+---PH,G¦bÿ+¦bÿPH,G¦bÿD bÿPH,G¦bÿD|bÿPH,G¦bÿD bÿPH,G¦bÿD bÿPH,G¦bÿD bÿPH,D bÿG¦bÿPH,) bÿG¦bÿPH,-£þÿ8¤þÿPH,©¢þÿô¢þÿPH,D þÿD þÿPH,D þÿD þÿPH,^ þÿPH,%d. PH, ype PH, wn tPH, nknoPH, !] UPH,. [PH,ptorPH,scriPH,y dePH,uritPH, SecPH, -PH,%lu PH,ow: PH,lu 1PH,h: %PH,higPH,es. PH, xpirPH, er EPH, NevPH, tSetPH, dLasPH, epwPH, tTimPH, ckouPH, floPH, ogofPH, astLPH, onlPH, tLogPH, lasPH, TimePH, wordPH, PassPH, badPH resPH tExpPH counPH acPH,set.PH,lue PH,o vaPH,IDNPH,ctGUPH,objePH,SIDPH,jectPH, obPH_%luPH_%lu PH. - PH.ed PH, nablPH, nt ePH, ccouPH, - aPH, d PH,ablePH, disPH,ountPH, accPH, -PH,ns: PH,ptioPH,nt oPH,ccouPH,+] APH,d

Badgers like LDAP!

It looks like the badger uploads PNG/image files to the C2 server. It also makes some queries to LDAP as well and will communicate with the Global Catalog. If it can't, it'll spit out some binding errors.

```
£þÿ8¤þÿPH,©¢þÿô¢þÿPH,D þÿD þÿPH,D þÿD þÿPH,^ þÿPH,%d.
                                                                                                                   PH, ype PH, wn tPH, nknoPH, !] UPH,.
[PH,ptorPH,scriPH,y dePH,uritPH, SecPH, -PH,%lu
PH,ow: PH,lu lPH,h: %PH,higPH,es.
PH, xpirPH, er
EPH, NevPH, tSetPH, dLasPH, epwPH, tTimPH, ckouPH, floPH, ogofPH, astLPH, onlPH, tLogPH, lasPH, TimePH, wordPH, PassPH, badPH, i
resPH,tExpPH,counPH,
acPH,set.PH,lue PH,o vaPH,IDNPH,ctGUPH,objePH,SIDPH,jectPH,
obPH %luPH %lu
PH. - PH.ed
PH,nablPH,nt ePH,ccouPH, - aPH,d
 PH.ablePH. disPH.ountPH. accPH.
                                   -PH.ns:
PH,ptioPH,nt oPH,ccouPH,+] APH,d
[PH,:%02PH,%02dPH,02d:PH,2d %PH,d-%0PH,-%02PH,%02dPH,at: PH,res PH,expiPH,ord PH,asswPH, - pPH,s
 PH,pirePH,r exPH,nevePH,ord PH,asswPH, - pPH,:
 PH,ingsPH,settPH,ire PH, expPH,wordPH,PassPH,[+] PH,athPH,ADsPPH,ls
PH.s
%PH,- %1PH,r PH,embePH,: mPH, %1sPH,[+]PH,1s:
PH,+] %PH,SE[PH,FALPH,TRUEP,P,P,P,PH,01x
PH, 0x%PH,] E:PH,
[-PH,iredPH,requPH,not PH,ing PH,atchPH,SI pPH,] AMPH,
[+PH,AMSIPH,tch PH,o paPH,le tPH,UnabPH,[-] PH,SI
PH,d AMPH,tchePH,] PaPH,
[+PH,ritePH,entWPH,twEvPH,ch EPH, patPH,e toPH,nablPH,-] UPH,e
[PH,WritPH,ventPH,EtwEPH,hed PH,PatcPH,[+] PH,ed
PH,atchPH,TW pPH,nd EPH,SI aPH,] AMPH,
[+PH,oundPH,ot fPH,v4 nPH,/v3/PH,R v2PH,] CLPH,
[-PH.0727PH.0.5PH.R v2PH.n CLPH.lu iPH.t v%PH.otnePH.ng dPH.unniPH.+] RPH.d
```

Searching lower down the list, we can see some of the information it collects, like Password Expiration, if the password never expires, and if there is a bad password supplied.

The Badger is Self Aware?

Continuing our string-hunt, here's one of the most interesting sets of strings... Badger itself is embedded as a string in the binary :facepalm:

```
WPH,%ls]PH.ed [PH,nectPH, ConPH,[+]PH,x%x
PH,E: 0PH,[-] PH,%lsPH,%ls\PH,%SPH,
FALSE
PH_STRUEPH_G%1PH_NNINPH_YRUPH_READPH_UEDPH_OUEPH_BLEDPH_DISAPH_OWNPH_UNKNP_PH_rAtoiPH_mpBadgePH_dgerWcscPH_trcm
pBaPH,BadgerSPH,erMemsetPH,cpyBadgPH,adgerMemPH,WcslenBPH,nBadgerPH,gerStrlePH,tchWBadPH,gerDispaPH,atchBadPH,d
gerDispPH,lsBaPH,+
%РН,----РН,----РН,----РН,----РН,----РН,----РН,----РН,----РН,----РН,----РН,----РН,----РН,----РН,----РН,----РН,----
PH.---PH.+---PH.ls
PH,1s %PH,%-20PH,--
PH, ----PH, ----PH, ----PH, ----PH, ----PH, ----PH, ----PH,
---PH, %1sPH,201sPH,
%-PH, %luPH,] E:PH,
[-PH, %lsPH,25lsPH,s %-PH,-201PH,th%PH,
PaPH,----PH,----PH,----PH,----PH,----PH,----PH,----PH,----PH,----PH,----PH,----PH,----PH,ls
-PH,ls %PH,%-25PH,0ls PH,%-2PH,aresPH,
ShPH, %lsPH,tingPH,meraPH, EnuPH,[+]PH,NamePH,ule PH,ModPH,NamePH,any PH,CompPH,ionPH,riptPH,DescPH,s
PH,: %1PH,51s PH, %-1PH,
```

I've already loaded up the binary into Ghidra and there's a whole lot of nothing. It seems to be a bit beyond my skill level to reverse engineer in a classic sense, so I'll have to do some more research on my own time to figure out if I can post a followup showing off the actual binary internals.

Misc Findings

Here are some interesting things I found that I wanted to include in the post, but couldn't easily write into the flow of the post. I still think this is worth mentioning.

PUNYCode! The thing I forgot existed?

Here is an interesting String Compare after executing a HTTP Request; it asppears that this badger is checking to see if some of the response headers contain xn--. This may be a sign that a threat actor is spoofing a common domain like <code>Google.com</code> to <code>http://xn--ggle-0nda.xn--om-ubc/</code>, which displays just like the normal domain does! Browser

settings can be configured to always display xn-, though some by default will render the link as normal. Thanks to <u>@ShitSecure</u> for pointing this out <3

65922	7:24:13.684 PM	6	KERNELBASE.dll	
65923	7:24:13.684 PM	6	KERNELBASE.dll	RtlFreeHeap (0x000000002710000, 0, 0x00000000027fcdd0)
65924	7:24:13.684 PM	6	KERNELBASE.dll	- RtlFreeHeap (0x000000002710000, 0, 0x0000000027a2070)
65925	7:24:13.684 PM	6	KERNELBASE.dll	RtIAllocateHeap (0x000000002710000, HEAP_CREATE_ENABLE_EXECUTE 1048576, 304)
65926	7:24:13.684 PM	6	KERNELBASE.dll	RtlFreeHeap (0x000000002710000, 0, 0x00000000044a1830)
65927	7:24:13.684 PM	6	crypt32.dll	wcschr ("inetsim.org", '*')
65928	7:24:13.684 PM	6	crypt32.dll	wcsstr ("inetsim.org", "xn")
65929	7:24:13.684 PM	6	crypt32.dll	-wcsstr ("159.65.186.50", "xn")
65930	7:24:13.684 PM	6	KERNELBASE.dll	- RtlFreeHeap (0x000000002710000, 0, 0x00000000044a1970)
65931	7:24:13.684 PM	6	KERNELBASE.dll	RtlFreeHeap (0x000000002710000, 0, 0x0000000027a8270)
65932	7:24:13.684 PM	6	WINTRUST.dll	malloc (240)

Traffic Generation to windowsupdate.com

Another interesting aspect of this badger is that it periodically reaches out to ctldl.windowsupdate.com. I originally thought this was Windows being Windows, but it turns out that this is hardcoded within the binary. This is likely a cloaking mechanism to throw off AV/EDR/Sandboxes.



I hope you all enjoyed :) ~Ronnie

Comments