[Guest Diary] Dissecting DarkGate: Modular Malware Delivery and Persistence as a Service.

dshield.org/diary/Guest Diary Dissecting DarkGate Modular Malware Delivery and Persistence as a Service/30700/

[This is a Guest Diary by John Moutos, an ISC intern as part of the SANS.edu Bachelor's Degree in Applied Cybersecurity (BACS) program [1].

Intro

From a handful of malware analysis communities I participate in, it is not uncommon for new or interesting samples to be shared, and for them to capture the attention of several members, myself included. In this case, what appeared to be a routine phishing PDF, led to the delivery of a much more suspicious MSI, signed with a valid code signing certificate, and with a surprisingly low signature-based detection rate on VirusTotal [2] (at time of analysis) due to use of several layered stages.

Context

Modern malware utilizing multiple layers of abstraction to avoid detection or response is not a new concept, and as a result of this continuous effort, automated malware triage systems and sandboxes have become crucial in responding to new or heavily protected samples, where static analysis methods have failed, or heuristic analysis checks have come back clean. Attackers are wise to this, and often use legitimate file formats outside of the PE family, or protect their final stage payload with multiple layers to avoid being detected through static analysis, and subsequently profiled through dynamic analysis or with the aid of a sandbox / automated triage system.

Analysis

The following sample not only fit the profile previously mentioned, but was also taking advantage of a presumably stolen or fraudulent code signing certificate to pass reputation checks.

At a first glance, the downloaded PDF appears normal and is of fairly small size.



Figure 1: Initial PDF Details

Opening the PDF with any suitable viewer, we can see an attempt to convince unknowing users to download a file, promising to resolve the fake load error.



Figure 2: Initial PDF Displayed

The "Open" button points to a wrapped doubleclick[.]net AD URL ("hxxps[://]adclick[.]g[.]doubleclick[.]net//pcs/click?f1587wub8-24-TzRtAOnedriveBskd&&adurl=//selectwendormo9tres[.]com? utm_content=AAhqplxaJo&session_id=3VHLBRuVfwDKTPWgylgR&id=b2WBu&filter=FSBMsIgzmQplvZl&lang=zh&locale=US"), which when followed arrives at "hxxp[://]95[.]164[.]63[.]54/documents/build-x64[.]zip/build-x64[.]msi". It is with this MSI where the initial infection chain starts, assuming the unsuspecting user proceeds to run the MSI after download. Inspecting the MSI, it does not appear to be artificially inflated with junk data as per the file size, and as a bonus it has a valid digital signature from a genuine certificate issued to "Inoellact EloubantTech Optimization Information Co., Ltd." from GlobalSign [3].

🖟 build-x64.n	nsi Pro	perties					×
Custom		Detai	s	F	Previous \	Versions	
General	Co	mpatibility	Digit	al Signatu	ures	Security	r -
1 6	build	x64.msi					
Type of file:	Windo	ows Installer P	ackage (.msi)			
Opens with:	ا 🔂	Vindows® inst	taller	Ch	ange		
Location:	C:\Us	ers\flare\Desl	ktop\Bad				
Size:	5.80 1	MB (6,090,752	bytes)				
Size on disk:	5.80 1	MB (6,090,752	bytes)				

Figure 3: Downloaded MSI Details

-					
Custom	Details	Previous Versions	Genera	Details Certification Path	
General Co	mpatibility Digi	tal Signatures Secur	ity		
· · · ·					
Signature list				ertificate Information	
Name of signer:	Digest algorithm	Timestamp	TT I	his certificate is intended for the following pur	pose(s):
Incellact Eloubar	n sha256	Monday, February 12,		• Ensures software came from software publisher	
		-		 Protects software from alteration after publication 	n
ital Signature Det	ails	?	×		
Advanced					
🖃 🔒 Digital S	ignature Informati	on		Refer to the certification authority's statement for det	talls.
📈 This digita	al signature is OK.			Iccued to: Incellact FloubantTech Ontimization 1	Information
This digita	al signature is OK.			Issued to: Inoellact EloubantTech Optimization I Co., Ltd.	Information
Signer information	al signature is OK.			Issued to: Inoellact EloubantTech Optimization : Co., Ltd. Issued by: GlobalSian GCC R45 EV CodeSigning	Information CA 2020
Signer information	n	ration Information Co	_	Issued to: Inoellact EloubantTech Optimization : Co., Ltd. Issued by: GlobalSign GCC R45 EV CodeSigning	Information CA 2020
Signer information	n ubantTech Optimiz	zation Information Co., Ltd.		Issued to: Incellact EloubantTech Optimization : Co., Ltd. Issued by: GlobalSign GCC R45 EV CodeSigning	Information CA 2020
Signer information Name: E-mail:	n ubantTech Optimiz Not available	zation Information Co., Ltd.		Issued to: Incellact EloubantTech Optimization : Co., Ltd. Issued by: GlobalSign GCC R45 EV CodeSigning Valid from 1/26/2024 to	Information CA 2020
This digita Signer information Name: E-mail: Signing time:	n Not available	ration Information Co., Ltd.		Issued to: Incellact EloubantTech Optimization : Co., Ltd. Issued by: GlobalSign GCC R45 EV CodeSigning Valid from 1/26/2024 to 1/26/2025	Information CA 2020
This digita Signer information Name: E-mail: Signing time:	ubantTech Optimiz Not available	ration Information Co., Ltd. y 12, 2024 7:54:27 AM		Issued to: Inoellact EloubantTech Optimization : Co., Ltd. Issued by: GlobalSign GCC R45 EV CodeSigning Valid from 1/26/2024 to 1/26/2025	Information CA 2020
This digita Signer information Name: E-mail: Signing time:	n JubantTech Optimiz Not available Monday, Februar	y 12, 2024 7:54:27 AM		Issued to: Incellact EloubantTech Optimization : Co., Ltd. Issued by: GlobalSign GCC R45 EV CodeSigning Valid from 1/26/2024 to 1/26/2025	Information CA 2020 uer Statement
This digita Signer information Name: E-mail: Signing time:	n ubantTech Optimiz Not available Monday, Februar	vation Information Co., Ltd. y 12, 2024 7:54:27 AM View Certificate		Issued to: Incellact EloubantTech Optimization : Co., Ltd. Issued by: GlobalSign GCC R45 EV CodeSigning Valid from 1/26/2024 to 1/26/2025 Install Certificate	Information CA 2020 uer Statement
This digita Signer information Name: E-mail: Signing time:	n ubantTech Optimiz Not available Monday, Februar	v 12, 2024 7:54:27 AM View Certificate		Issued to: Inoellact EloubantTech Optimization I Co., Ltd. Issued by: GlobalSign GCC R45 EV CodeSigning Valid from 1/26/2024 to 1/26/2025 Install Certificate	Information CA 2020 uer Statement
This digita Signer information Name: E-mail: Signing time: Countersignature Name of signer	I signature is OK. UbantTech Optimiz Not available Monday, Februar	vation Information Co., Ltd. y 12, 2024 7:54:27 AM View Certificate Timestamp		Issued to: Incellact EloubantTech Optimization I Co., Ltd. Issued by: GlobalSign GCC R45 EV CodeSigning Valid from 1/26/2024 to 1/26/2025 Install Certificate	Information CA 2020 uer Statement

Figure 4: MSI Signature & Certificate Details

To extract the content from the MSI, there are a plethora of tools that can be used. Universal Extractor [4], 7-Zip [5], and the built-in extractor feature in the multi-purpose analysis tool "Detect It Easy" (DIE) [6] will handle the job without issue.

File name					
> C:\Users\flare\De	sktop\Bad\build-x64	.msi			
File type Binary 🔹	File size 5.81 MiB		Base address 00000000	Entry point	00000 >
File info MIME	Memory map	Disasm	Hex Search	Strings Signatu Hash Entrop	virusTotal VirusTotal
Binary					
Scan		Endianness	Mode	Architecture	Туре
Automatic	-	LE	Unknown	NOEXEC	Unknown
✓ Binary Installer: Micro Data: Microsof	osoft Installer(MSI) ft Compound				S ? S ?

Figure 5: MSI Opened in DIE

With the content of the MSI extracted, there are two important files to note, the first named "Binary.bz.WrappedSetupProgram", which is the embedded cabinet (CAB) file, and the second named "Binary.bz.CustomActionDII" which is an embedded DLL.

File name						
> C:\Users\flare\De	esktop\Bad\build-x64	\Binary.bz.Wra	appedSetupProgram			
File type	File size		Base address	Enti	ry point	
Binary 🔻	5.55 MiB		00000000		00000000	>
File info	Memory map	Disasm	Hex	Strings	Signatures	VirusTotal
MIME			Search	Hash	Entropy	Extractor
Binary						
Scan		Endianness	Mode	Architec	ture	Туре
Automatic	-	LE	Unknown	NOEXE	ic 🖉	Unknown
- Rinan/						
Archive: Micro	osoft Cabinet File(1	.03)[100.0% 3	files]			S ?
Data: CAB arc	hive	10070101010101010				S ?

Figure 6: Extracted Cabinet File in DIE

File name				
> C:\Users\flare\Deskt	top\Bad\build-x64\Binary.bz.Cu	stomActionDll		
File type File T	ile size 208.00 KiB	Base address 10000000	Entry point 10010d4b	
File info	Memory map Disasm	Hex Strings	Signatures	VirusTotal
MIME		Search Hash	Entropy	Extractor
PE	Export Import	Resources .NET	TLS	Overlay
Sections	Time date stamp	Size of image	Resources	
0005 >	2022-07-23 05:01:25	00039000	Manifest	Version
Scan	Endianness	Mode Arc	hitecture	Type
Automatic				Ditt
 PE32 Compiler: EP:Mic Compiler: Micros Linker: Microsoft 	crosoft Visual C/C+ + (2008-20 soft Visual C/C+ + (2008 SP1)[: Linker(9.0)[DLL32]	010)[DLL32] libcmt]		S? S? S?

Figure 7: Extracted DLL File in DIE

The DLL only serves to assist in the deployment of the cabinet file during the MSI installation process, but it should be noted it also has several other execution paths, corresponding to different installer modes and the respective entry point followed.

🛃 Choose an entry point			×
Name	Address	Ordinal	
f CheckReboot(x)	1000A5D0	1	
f InstallFinish1(x)	1000A510	2	
f InstallFinish2(x)	1000A740	3	
<u>f</u> InstallMain(x)	1000A9D0	4	
f InstallPrepare(x)	1000A4A0	5	
f InstallRollback(x)	1000ABC0	6	
SubstWrappedArguments(x)	1000AC80	7	
f UninstallFinish1(x)	1000B280	8	
<u>f</u> UninstallFinish2(x)	1000B6E0	9	
f UninstallPrepare(x)	1000AC90	10	
📝 DllEntryPoint	10010D4B	[main entry]	

Figure 8: Extracted DLL Entry points

Returning back to the extracted cabinet (CAB) file, we can simply open it with 7-Zip to view the contents.

C:\Users\flare\De	C:\Users\flare\Desktop\Bad\build-x64\Binary.bz.WrappedSetupProgram\							
File Edit View Fi	avorites	Tools	Help					
4 - V	•	-	×	ñ				
Add Extract Test	Сору	Move	Delete	Info				
C:\Users\fla	re\Deskto	op\Bad\b	uild-x64\	Binary.bz.Wrapped	SetupProgram\			
Name			Size	Modified	Attributes	Method		
CoreFoundation.d	II .	3	756 032	2024-02-12 07:46	А	None		
📧 iTunesHelper.exe			366 944	2024-02-12 07:46	А	None		
🗟 sqlite3.dll		1	699 957	2024-02-12 07:46	Α	None		

Figure 9: Cabinet File Contents

The file "iTunesHelper.exe" has a valid signature from Apple, whereas the "sqlite3.dll" and "CoreFoundation.dll" files are unsigned. These files will presumably be loaded ("CoreFoundation.dll" is listed in the Import Table) when "iTunesHelper.exe" is launched, so I will focus on these files.

Due to how Windows searches for and loads DLLs [7], the "iTunesHelper" application will load any DLL named "CoreFoundation". Windows first searches the directory where the application launched from, and in this case, it would find a match and load the DLL. Windows then falls back to the System32 directory, then the System directory, the Windows directory, the current working directory, all directories in the system PATH environment variable and lastly all directories in the user PATH environment variable.

ITunesHelper.exe P		
Security	Details	Previous Versions
General	Compatibility	Digital Signatures
Signature list		
Name of signer:	Digest algorithm	Timestamp
Apple Inc.	sha1	Friday, May 12, 2023
Apple Inc.	sha256	Friday, May 12, 2023
Digital Signature Deta	ils	?
General Advanced		
Digital Signature	gnature Informatio signature is OK.	n
Digital Signer information	gnature Informatio signature is OK.	n
Signer information Name:	gnature Informatio signature is OK. Apple Inc.	n
Signer information Name: E-mail:	gnature Informatio signature is OK. Apple Inc. Not available	n
Signer information Name: E-mail: Signing time:	gnature Informatio signature is OK. Apple Inc. Not available Friday, May 12, 20	n 23 2:08:29 PM
Digital Sig This digital Signer information Name: E-mail: Signing time:	gnature Informatio signature is OK. Apple Inc. Not available Friday, May 12, 20	n 223 2:08:29 PM View Certificate
Digital Sig This digital Signer information Name: E-mail: Signing time:	gnature Informatio signature is OK. Apple Inc. Not available Friday, May 12, 20	n 123 2:08:29 PM View Certificate
Digital Sig This digital Signer information Name: E-mail: Signing time: Countersignatures Name of signer:	gnature Informatio signature is OK. Apple Inc. Not available Friday, May 12, 20 E-mail address:	n 223 2:08:29 PM View Certificate

Figure 10: iTunesHelper EXE Signature

File n	ame C: \Users\flare \Desk	ktop \Bad \build-x64\Bi	nary.bz\jTunesHelp	per.exe					
							_		<
		Hex	Disasm	Strings	Memory map	Entropy	Heuristic scan	✓ Reador	ıly
164		Hash 32					Г		_
0000	0057c201bd34	c011e49a					L	Save	
# (DriginalFirstThunk	TimeDateStamp F	orwarderChain	Name	FirstThunk	Hash	Name		
0	0003d310	00000000	00000000	0003d89e	0002e058	1af29737	CoreFoundatio	n.dll	
1	0003d720	00000000	00000000	0003d97c	0002e468	5ee47df8	SETUPAPI.dll		
2	0003d760	00000000	00000000	0003d9ae	0002e4a8	cd5b2a82	SHLWAPI.dll		
3	0003d340	00000000	00000000	0003dd84	0002e088	9b1995b8	KERNEL32.dll		
4	0003d778	00000000	00000000	0003de76	0002e4c0	19ffe8be	USER32.dll		
5	0003d2b8	00000000	00000000	0003df2a	0002e000	c12ff91a	ADVAPI32.dll		
6	0003d7f0	00000000	00000000	0003dff8	0002e538	ab5f61b5	ole32.dll		
7	0003d6e0	00000000	00000000	0003e002	0002e428	f32aa809	OLEAUT32.dll		

Figure 11: iTunesHelper EXE Import Table

Upon closer inspection at the "sqlite3" DLL, it does not appear to be a valid PE (Portable Executable) file, but it will be revisited.

File name		
> C:\Users\flare\	Desktop\Bad\build-x64\Binary.bz\sqlite3.dll	
File type	🖸 Hex — 🗆	×
Binary 🔻		
	Data inspector	Readonly
File info	Hex	
NATING	7 08 09 0a 0b 0c 0d 0e 0f Symbols	
MIME	5 56 23 10 cf a2 5a 12 ceSZEV#Z	· · · 📙 🖵
	a 86 b9 58 4c 4b 53 5a 45 .2]L.WZXLKS	ZE
Binary	5 56 7a 58 4c 4b 53 5a 45 VzXLKSZEVzXLKS	ZE
	5 56 7a 58 4c 4b 52 5a 45 VzXLKSZEVzXLKR	ZE
	8 77 c2 59 00 86 72 ca d5 .jXBT.S.w.Yr	
	a 31 08 39 21 6b 3e 2f 361?k#(*1.9!k>	/6
	b 76 Of 36 28 2e 21 7a 12 "Z:)k!/+v.6(.!	z.
Scan	2 56 7a 58 4c 4b 53 5a 45 ?.nxFY~rVzXLKS	ZE
Automatic	5 56 7a 58 4c 4b 53 5a 45 VzXLKSZEVzXLKS	ZE Ur
	5 56 7a 58 4c 4b 53 5a 45 VzXLKSZEVzXLKS	ZE
Binary	5 56 7a 58 4c 4b 53 5a 45 VzXLKSZEVzXLKS	ZE
	5 56 7a 58 4c 4b 53 5a 45 VzXLKSZEVzXLKS	ZE
	5 56 7a 58 4c 4b 53 5a 45 VzXLKSZEVzXLKS	ZE
	5 56 7a 58 4c 4b 53 5a 45 VzXLKSZEVzXLKS	ZE
		•
	Cursor:000000000000000 Selection:000000000000000000000000000000000000	000000
Signatures ✓ R		ose —

Figure 12: sqlite3 File Junk Data

Inspecting the "CoreFoundation" DLL with a disassembler such as IDA [8], Ghidra [9], or Binary Ninja [10], and going to the main entry point, we can trace the execution flow up to where a function named "CFAbsoluteTimeAddGregorianUnits" is called, which when followed checks if the process it has been loaded into is running from the path "c:\\debug", followed by a message box popup with the string "debug dll start". This functionality is unrelated to the malicious behavior, but is a good indication the file has been tampered with, along with the lack of a valid signature.

DllEntryPoint	proc near			DATA XREF: HEAD
				.pdata:0000000
var_s20 var_s30 var_s3C var_s40 var_s48 var_s17C	<pre>= qword ptr = qword ptr = dword ptr = qword ptr = byte ptr = dword ptr</pre>	20h 30h 3Ch 40h 48h 17Ch		
;unwind { //	sub_40E440pushrbjsubrsjmovrbjmov[rlmov[rlmov[rlmov[rlmov[rlmov[rlmov[rlmov[rlmov[rlmov[rltesteaxjnzsholeardxmovraxmovraxmovraxmovraxmovraxmovraxmov[rax	<pre>b, 180h b, rsp bp+var_s30], bp+var_s30], bp+var_s30], bp+var_s40], c, [rbp+var_s40], c, [rbp+var_s40], c, 2 c, 2 c, 2 c, 2 c, 2 c, 2 c, 2 c,</pre>	rc: ed: r8 s48 s48 y e 09 s48 30 30 s3C s3C ra: s40	x x] cx]] x GregorianUnits

Figure 13: CoreFoundation DLL Entry Point



Figure 14: CoreFoundation DLL Debug Directory Check

Following the "CFAbsoluteTimeAddGregorianUnits" execution flow further down, we can find a reference to the bundled "sqlite3" DLL.



Figure 15: sqlite3 File Reference in CoreFoundation DLL

Switching back to the "sqlite3" DLL, using DIE to view the strings in the file, there appears to be an Autolt compiled script header value denoted by the characters "AU3!EA06". Opening the the file with a hex editor such as HxD [11] or DIE (DIE has a built-in one), we can confirm the presence of the Autolt [12] compiled script header. This will be revisited shortly.

							_		
🔹 sqlite3.dll									
Offset(h)	00	01	02	03	04	05	06	07	Decoded text
001288C0	4B	BE	98	6C	4A	A9	99	4C	K¾~1J©™L
001288C8	53	0A	86	D6	48	7D	41	55	S.tÖH} <mark>AU</mark>
001288D0	33	21	45	41	30	36	4D	A8	3 ! EA0 6M"
001288D8	FF	73	24	Α7	3C	F6	7A	12	ÿs\$§<öz.
00120020	F1	67	20	C1	0.0	57	CD	40	for AnaleC
Figure 16:	Aut	tolt	Co	mp	ilec	l Sc	rip	t He	eader in solite3

Switching gears back to the "CoreFoundation" DLL, following the references to the "sqlite3" DLL, we can find a block of code that resembles a XOR decryption routine. Looking for cross-references to this decryption code leads to more references to the "sqlite3" file, along with a

familiar string. The string "VzXLKSZE" is scattered throughout the "sqlite3" file, and fills up the majority of the space within the file. Between this, and the reference to the XOR decryption routine, we can assume this may be the key used to decrypt the "sqite3" file.



Figure 17: sqlite3 File and Key References in CoreFoundation DLL

📓 sqlite3.dll									
Offset(h)	00	01	02	03	04	05	06	07	Decoded text
00000000	1B	20	1D	1E	A3	53	5A	45	£SZE
0000008	56	23	10	CF	A 2	5A	12	CE	V#.Ï¢Z.Î
00000010	97	32	5D	4C	AB	57	5A	BA	—2]L«WZ°
00000018	86	В9	58	4C	4B	53	5A	45	† ¹ XLKSZE
00000020	56	7A	58	4C	4B	53	5A	45	VzXLKSZE
00000028	56	7A	58	4C	4B	53	5A	45	VzXLKSZE
00000030	56	7A	58	4C	4B	53	5A	45	VzXLKSZE
00000038	56	7A	58	4C	4B	52	5A	45	VzXLKRZE
0000040		<u>кл</u>	5.9	42	54 140	도 7 5 도:	52	88	1 IVRTOS*

Figure 18: XOR Key in sqlite3 File

Loading "sqlite3" into a tool like CyberChef [13], the XOR operation can be used, and when provided with the discovered key, the file content is decrypted, and appears to have a valid PE header, denoted by the MZ characters at the beginning.



Figure 19: XOR Decrypting sqlite3 File

After saving the decrypted content ("sqlite3decrypted.dll") to disk, we can load it into DIE to verify it does resemble a valid PE file.



Figure 20: Decrypted sqlite3 File in DIE

Dropping the decrypted binary ("sqlite3decrypted.dll") into a disassembler and following execution flow from the entry point, we can see the next stage takes the form of the Autolt compiled script discovered before, and this DLL serves to drop the script, the actual Autolt executable, and a "test.txt" file into the "c:\temp" directory, before executing the script with Autolt.



Figure 21: Decrypted sqlite3 File Pseudocode

To extract the compiled script, we can revisit the original encrypted "sqlite3.dll" file, and look for the delimiter used to separate the script content from the rest of the binary. It should also be noted that the delimiter string "delimitador" can be found in the "sqlite3decrypted.dll" file.



Figure 22: Delimiter String in Decrypted sqlite3 File

Knowing the string delimiter to look for, we can carve out the Autolt compiled script from the original "sqlite3" file. A hex editor can be used to do this easily.

8 s	qlite	3.dll	F.D.	sql	ite3d	lecry	pted	.dll								
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	OF	Decoded text
21	5A	45	64	65	6C	69	6D	69	74	61	64	6F	72	A3	48	!ZEdelimitador <mark>£</mark> H
4B	BE	98	6C	4A	Α9	99	4C	53	0A	86	D6	48	7D	41	55	K¾~1J©™LS.†ÖH}AU
33	21	45	41	30	36	4D	A 8	FF	73	24	A 7	3C	F6	7A	12	3!EA06M″ÿs\$§<öz.
F1	67	\mathbf{AC}	C1	93	E7	6B	43	CA	52	A6	AD	00	00	El	BB	ñg−Á"çkCÊR¦á»
3A	21	A5	29	E3	EC	E7	0B	98	2E	40	BD	El	9A	DE	80	:!¥)ãìç.~.@≒ášÞ€
46	B1	9D	6B	3B	21	D4	B1	D6	75	ЗA	C8	ЗD	C6	DO	33	F±.k;!Ô±Öu:È=ÆÐ3
F7	14	AF	СВ	17	A2	94	01	8D	13	88	FE	64	95	61	E7	÷.⊤Ë.¢″^þd•aç
B6	4D	1 🛛	F8	00	00	OD	D5	FD	C4	2B	1 F	97	4D	1F	17	9TM ด Õiä+—M
Fio	iure	e 23	: S	tart	De	lim	iter	' in	Ori	ain	al s	alit	te3	File)	

	S S	qlite	3.dll	FD	sql	ite3d	lecry	pted	.dll								
	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	OF	Decoded text
	58	58	58	58	58	58	58	58	58	58	58	58	58	58	58	58	XXXXXXXXXXXXXXXXX
	58	58	58	58	58	58	58	58	58	58	58	58	58	58	58	58	xxxxxxxxxxxxxxx
	58	58	58	58	58	58	58	58	58	58	58	58	58	58	58	58	*****
	58	58	58	58	58	58	58	58	58	58	58	58	58	58	58	58	*****
	58	58	58	58	58	58	58	58	58	58	58	58	58	58	58	58	xxxxxxxxxxxxxxxxx
	58	58	58	58	58	58	56	7A	58	4C	4B	53	5A	45	64	65	XXXXXXVzXLKSZE <mark>de</mark>
	6C	69	6D	69	74	61	64	6F	72	28	6E	71	5D	4E	2A	30	limitador(nq]N*0
	43	56	33	26	52	65	4D	4F	74	4A	7D	55	61	44	7B	57	CV3&ReMOtJ}UaD{W
F	Fig	ure	24:	Er	nd E	Deli	mit	er i	n O	rigi	inal	sq	lite	3 F	ile		

The Autolt script, now saved to disk, unfortunately is unusable while still compiled, and must be decompiled with a tool such as myAutToExe [14].

🔝 script.a3x	FD S	qlite	3.dll	FD	sql	ite3d	ecry	pted	.dll								
Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	OF	Decoded text
00000000	AЗ	48	4B	BE	98	6C	4A	Α9	99	4C	53	0A	86	D6	48	7D	£HK¾~1J©™LS <mark>.</mark> †ÖH}
00000010	41	55	33	21	45	41	30	36	4D	8 A	FF	73	24	A 7	3C	F6	AU3!EA06M¨ÿs\$§<ö
00000020	7A	12	F1	67	AC	C1	93	E7	6B	43	CA	52	Α6	AD	00	00	z.ñg¬Á"çkCÊR¦
00000030	E1	BB	ЗA	21	A 5	29	E3	EC	E7	0B	98	2E	40	BD	E1	9A	á»:!¥)ãìç.″.@⊁≾áš
00000040	DE	80	46	B1	9D	6B	3B	21	D4	B1	D6	75	ЗA	C8	ЗD	C6	Þ€F±.k;!Ô±Öu:È=Æ
00000050	DO	33	F7	14	AF	СВ	17	A2	94	01	8D	13	88	FE	64	95	Ð3÷. Ë.¢″^þd∙

Figure 25: Compiled Autolt Script Extracted

With the script decompiled, we can see it is obfuscated using character substitution, which we must reverse before we can proceed.

script	ː_restore.au3 ⊠ test.txt ⊠
1	#NoTrayIcon
2	GUICREATE("xtzqkgbyf",135,902)
3	<pre>\$A=STRINGSPLIT(FILEREAD(@SCRIPTDIR&"\test.txt"),"",2)</pre>
4	\$BZXRGJFO=\$A[58]&\$A[6]&\$A[62]&\$A[58]&\$A[48]&\$A[58]&\$A[
5	\$BZXRGJF0&=\$A[6]&\$A[6]&\$A[71]&\$A[64]&\$A[53]&\$A[27]&\$A[
	\$BZXRGJF0&=\$A[53]&\$A[64]&\$A[53]&\$A[67]&\$A[71]&\$A[6]&\$A
7	\$ <i>BZXRGJF0</i> &=\$A[52]&\$A[67]&\$A[52]&\$A[37]&\$A[52]&\$A[64]&\$
	[52]&\$A[58]&\$A[71]&\$A[71]&\$A[52]&\$A[27]

Figure 26: Decompiled Autolt Script Obfuscation

The Autolt "STRINGSPLIT" function [15] is being called on the content of test.txt, read using "FILEREAD" [16], with a blank delimiter, and with mode 2, which sets the starting count of the array to 0 instead of 1.



Figure 27: test.txt File Content

For example; \$A[0] would be the character "(", and \$A[1] would be the character "n".

Once the character substitution is reversed and the script is now readable, we can see it construct shellcode from the content above and attempt to load and execute it in memory. It additionally checks if any Sophos products are installed, and will switch execution flows if this check fails.

The VirtualProtect Windows API [17] is used to modify the allocated memory region protection, so the shellcode can be copied and executed using the EnumWindows Windows API [18].



Figure 28: Autolt Script Content

Following the reference to the shellcode data stored across the variable named "\$BZXRGFO", we can see that it uses the Autolt function BinaryToString [19], which converts a given value from binary representation to string form.

Knowing this we can extract the embedded shellcode blob and hex decode it. Once again, CyberChef has a hex decode operation that can handle this task for us.

Recipe		Input	+ 🗆 🔁 🔳 🖬
From Hex Delimiter Auto	0 11	90E989030000007458414F794278545A70496E4E54436A61644F686C4B546977684D4E6 C6455754B79566750774E4975784C63556A525247614F6C614D684E756E5977646A4260 44436F4F4D7549645672766159585563477474544F71527063464A464351506F5147787 27562676D0844744E64546A78534D754762666E4A594C6C4454426F454F574F4D0654671 7258636D4C645152674C45666E427857526878577368596C51784B5968445A444458684 263484F484B6D706D517471444C4A66664D685A446474586F59524D796766434F4A4254 7271596762466470656F6573597250415861474A72566E634F6C7276797257745A48656 D646C71484579646D4C7878494F4F584B6F58424B7964496D5979664D4852714978416F 6E7671776D7A6A43486465454F496F4A6A4C576574B70657A574659714E4734965596 66876766F446677442427763355564694D5953D722066584594F7162725863737A4D 45253454F7A73696F49464D477563544F677664B464259606A654F71714F46486C594 Siz	File details
		<pre>me 91976 = 1 Output O LvvZIoZrSVxEQcFyreJgvOcjtvRMEcwzhCakvmHAzPttVlvTYtIeIvQojTIMszWStJljXKdUxHpFa FcYauBxahIrjEipehBBLXrMOXNdySIfccmgyILNSAbdpynNAIgJzqhnMqWneNTTbdmChBwFycMAuP cxbtDYquVWHfldnjpgamaisfnsveahYysxQqnKNWDNRPMdsHtBsHxkgZwURdOMjGWSXfKHskWSKXLW UYcpYADLcYuihYeuxuQpmUVEdZxhaasKoGMyfbiiOevsyzJiTzqZcgMnbkUPNNylPnNvMXSUbwNSc qEFbwukbsnEYpRpXDlBWxsUzukukGJbEQMNbDGxnXNDobzsUmFJdoulSGQkFPSkTomFTsdLCLK0Gq X+è PVVV9ALcYuihYeuxuQpmUVEdZxhaasKoGMyfbiiOevsyzJiTzqZcgMnbkUPNNylPnNvMXSUbwNSc gEFbwukbsnEYpRpXDlBWxsUzukukGJbEQMNbDGxnXNDobzsUmFJdoulSGQkFPSkTomFTsdLLCWOGq X+è PVVV9ALcYuihYeuxuQpmUVEdZxhaasKoGMyfbiiOevsyzJiTzqZcgMnbkUPNNylPnNvMXSUbwNSc gFFbwukbsnEYpRpXDlBWxsUzukukGJbEQMNbDGxnXNDobzsUmFJdoulSGQkFPSkTomFTsdLLCWOGq X+è PVVV9ALcYuihYeuxuAghtgytytytytytytytytytytytytytytytytytyt</pre>	Tr Raw Bytes ← IWZnyMuvKFhdJtkpUjJ 'bQGWJMWZgVCMXXUckQN ItFNAqtPlOvjIFNSVvJI VfbsThJCpRrfprmiofx IPFSkyNlwIVgMZER&\\ This program must
STEP 💆 BAKE!	Luto Baka	۱۰٬۵۰٬۰٬۵٬۰٬۰٬۵٬۰٬۰٬۵٬۰٬۰٬۵٬۰٬۰٬۰٬۵٬۰٬۰٬۵٬۰	·/////////////////////////////////////

Figure 29: Decoding the Included Shellcode

After saving the decoded shellcode data as a file, if we open it with a hex editor, we can see the start of a valid PE header after a large chunk of garbage data. To properly disassemble the file with a tool such as IDA or Ghidra, the garbage data will need to be removed (if the junk data is

left, the entry point will have to be manually specified).

📓 sqlite3.dll	ao f	inals	tage	.dat													
Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	OF	Decoded text
00000B10	6F	62	7A	73	55	6D	46	4A	64	6F	75	6C	53	47	51	6B	obzsUmFJdoulSGQk
00000B20	46	50	53	6B	54	6F	6D	46	54	73	64	43	UHA	6 3	a	a	FPSkTomFTsdCLCWO
00000B30	47	71	50	46	53	6B	79	4E	6C	77	49	56	67	4D	5A	45	GqPFSkyNlwIVgMZE
00000B40	52	E8	00	00	00	00	58	83	E8	09	50	05	00	AO	00	00	RèXfè.P
00000B50	FF	DO	C3	00	00	40	00	1A	00	00	00	00	00	00	00	00	ÿÐÃ@.
00000B60	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	L
00000B70	00	00	00	00	00	00	00	00	00	00	01	00	00	BA	10	00	· · · · · · · · · · · · · · · · · · ·
00000B80	0E	1F	Β4	09	CD	21	B8	01	4C	CD	21	90	90	54	68	69	.Í!LÍ!Thi
00000B90	73	20	70	72	6F	67	72	61	61	25	ARE	a(lei	<mark>רק</mark>	tai	7 [2	s program must b
00000BA0	65	20	72	75	6E	20	75	6E	64	65	72	20	57	69	6E	33	e run under Win3
00000BB0	32	0D	0A	24	37	00	00	00	00	00	00	00	00	00	00	00	2\$7
E			!							1 111	_						

Figure 30: PE Header in Extracted Shellcode File

The junk data can be stripped with a hex editor or other file manipulation tools, and once removed we can load the cleaned file into DIE to verify the file is detected as a valid PE.

File name			
> C:\Users\flare\Desktop\Bad\build->	k64\Binary.bz\temp\fi	nalstage.dat	
File type File size	КıВ		
Scan	Endianness	Mode	Architecture
Automatic 🔻	LE	32-bit	1386
 PE32 Compiler: Borland Delphi(6-7 Linker: Turbo Linker(2.25*,Delp Overlay: Binary 	or 2005)[-] phi)[GUI32]		

Figure 31: Extracted Shellcode File in DIE

Loading this final stage file into a disassembler, and going to the entry point, we can spot the XOR key utilized in previous stages

start:		
	push	ebp
	mov	ebp, esp
	mov	ecx, 4
loc_402A44:		; CODE XREF: CODE:00402A49↓j
	push	
	push	
	dec	ecx
	jnz	short loc_402A44
	push	ecx
	mov	eax, offset dword_402A0C
	call	<pre>@Sysinit@@InitExe\$qqrpv ; Sysinit::linkproc InitExe(void *)</pre>
	xor	eax, eax
	push	ebp
	push	offset loc_402B8F
	push	dword ptr fs:[eax]
	mov	fs:[eax], esp
	mov	eax, offset dword_404680
	mov	edx, offset aVzxlksze ; "VzXLKSZE"
	call	sub_4016A0
	mov	edx, offset dword_404684
	mov	eax, 1
	call	<pre>@System@ParamStr ; System::ParamStr</pre>
	cmp	ds:dword_404684, 0
	jnz	short loc_402A9E
	push	

Figure 32: Final Stage File Disassembly

With the help of a debugger (I used x32dbg [20]), we can dump the final stage config data at runtime post-decryption to reveal the C2 server it reports home to, which is located at the domain "prodomainnameeforappru[.]com (46.21.157.142)". It should be noted that the final stage shellcode when executed in memory at runtime, will be mapped in a newly spawned "VBC.exe" (Visual Basic command line compiler) process.

<u>File V</u> iew I	<u>D</u> ebug Traci <u>ng P</u> lug	jins Favour <u>i</u> tes <u>O</u> pt	tions <u>H</u> elp May	8 2021 (TitanEngi	ne)								
📄 🥑 🔳	🔿 II 🕴 🖓 👒	🖢 🎍 🛊 🔹 📓	🥖 🗏 🛷 🥒	fx # A2	1								
CPU	Log 🖺 Notes	Breakpoints	Memory Map	Call Stack	📆 SEH	Script	🐏 Symbols	<> Source	₽ R	eferences	🐭 Threads	晶 Handles	
EIP ECX	7773334C 7773334F	C2 2000 90	nop						^		Hide	FPU	
	 77733350 77733355 7773335A 7773335C 	B8 8D000000 BA <u>908A7477</u> FFD2 C2 1000	mov eax mov edx call ec ret 10	(,8D (,ntdll.77748 dx	A90		ZwAlpcSetI	nformation		EAX EBX ECX	00000000 04B8FE20 7773334C	ntd11.77733	34C
	 7773335F 77733360 77733365 7773336A 7773336C 7773336E 	90 B8 8E000500 BA <u>908A7477</u> FFD2 C2 0800 90	nop mov eax mov edx call ec ret 8	(,5008E (,ntdll.77748 1x	A90		ZwAr eMappe	dFilesTheSam	1e	EDX EBP ESP ESI EDI	00000000 04939358 049392D4 04BE2808 04B8FED8	"\\ŒñvÀ\x03 L"Px\x02"	
	 77733370 77733375 7773337A 7773337C 	B8 8F000800 BA <u>908A7477</u> FFD2 C2 0800	mov eax mov edx call ec ret 8	(,8008F (,ntdll.77748 dx	A90		ZwAssignPr	ocessToJobOb)jec	EIP EFLAGS	7773334C 00000304	ntd11.77733	34C
	 7773337F 77733380 77733385 7773338A 	90 B8 90000000 BA <u>908A7477</u> FFD2	mov eax mov edx call ec	(,90 (,ntdll.77748 dx	A90		ZwAssociat	eWaitComplet	ior	OF 0 CF 0	F 1 AF 0 SF 0 DF 0 TF 1 IF 1		
	 7773338C 7773338F 77733390 77733395 	C2 2000 90 B8 91000000 BA <u>908A7477</u>	ret 20 nop mov eax mov ed	(,91 (,ntdll.77748	A90		ZwCallEncl	ave		LastEr LastSt	ror 00000000 atus 00000000 B ES 0053) (ERROR_SUC)) (STATUS_SU	CESS
	 7773339A 7773339C 7773339F 	FFD2 C2 1000 90	call ec ret 10 nop	ix						ES 002	B DS 002B		>
	• 777333A0 • 777333A5	B8 92000000 BA <u>908A7477</u>	mov eax mov edx	(,92 (,ntdll.77748	A90		NtCancelIo	FileEx	~	Default (s	tdcall)	▼ <u>5</u> 🗘 🗆 U	Inlocked
20 1 1	<								>	2: [esp	+8] 40020000	L "Bx\ x02"	î
.text:77733	334C ntdll.dll:\$7	334C #7274C								4: [esp 5: fesr <	04822808 0+10] 0488FED 0+14] 048F280	8 8 "Px\x02"	>
Ump 1	Dump 2	Dump 3 🛛 💷 Dump 4	Dump 5	🥘 Watch 1	[x=] Locals	Struct	049393	1C 777E4FD8	ntdl	1.777E4F	D8		^
Address He	ex			ASCII			049393	24 00000078					
048938E0 DE	D CC BB AA 02 00	00 00 00 00 00 0	0 00 00 00 00	ÝÌ»ª			049393	28 04939348 2C 04B8FE20					
04893C00 F	F FF FF FF 35 00	00 00 00 00 00 00 0	0 00 00 00 00	ÿÿÿÿÿ5			049393	30 04B8FC20	L"pre	odomainn	ameeforappru	. com"	-
04B93C10 00 04B93C20 C0	0 00 00 00 00 00 00	00 00 35 00 09 0 00 46 40 90 05 2	0 00 00 1C 00 2 9E 7E CF 11	À	~î.		049393	38 00000036	L pr				
04893C30 A	E 5A 00 AA 00 A7	11 2B 00 00 00 0	0 00 00 00 00	•Z.•.§.+	····		049393	3C 7530D42A 40 049393E4	dnsa	p1.7530D	42A		
04893C40 /0	E 00 6E 00 6F 00	6D 00 65 00 65 0	0 66 00 6F 00	n.n.a.m.e.e.	d. 1. f. 0.		049393	44 04BE2870					
04B93C60 72 04B93C70 66	2 00 61 00 70 00 F 00 6D 00 00 00	70 00 <u>72 00 75 0</u> 00 00 00 00 00 0	0 2E 00 63 00 0 00 00 00 00	r.a.p.p.r.u. o.m	c.		▼ 049393 049393 049393	4C 76F1CE27	retu	rn to rp	crt4.76F1CE2	7 from rpcrt	4.70
							< 1						1

Figure 33: Extracting C2 Domain with x32dbg

Flow Summary

- Initial PDF ("case_-2023_4824647818.pdf"): Deliver MSI via AD download link.
- Downloaded First Stage MSI ("build-x64.msi"): Unpack embedded cabinet file.
- Extracted Cabinet File ("Binary.bz.WrappedSetupProgram"): Contains encrypted next stage DLL, and dummy app to use with tampered DLL for sideloading.
- Dummy App ("iTunesHelper.exe"): Used to load tampered import DLL.
- Tampered Import DLL ("CoreFoundation.dll"): Used to load and XOR decrypt next stage DLL
- Encrypted Second Stage DLL ("sqlite3.dll"): Drop embedded compiled Autolt script, Autolt binary, and character substitution alphabet, and invoke compiled script with Autolt binary.
- Autolt Binary ("autoit.exe"): Used to execute compiled Autolt script.
- Character Substitution Alphabet ("test.txt"): Used to run compiled Autolt script (or deobfuscate a decompiled version).
- Compiled Third Stage Autolt Script ("script.a3x"): Construct final stage shellcode to load and execute in allocated memory.
- Final Stage DarkGate Agent ("finalstage.dat" or found in memory of host "vbc.exe" process at runtime): Beacon home and provide remote access / additional malware delivery functionality.

Takeaway

DarkGate is a commodity loader with remote access and modular plugin capability, written in Borland Delphi that is advertised under the Malware-as-a-Service (MaaS) business model on popular cybercrime forums [22]. It mainly serves to deliver other malware, commonly infostealers to compromised hosts and either aid in exfiltration of the data or futher access and persistence. As modern AV/EDR products scrutinize PE files much more aggressively, alternative file types that can nest additional stages and still look legitimate are becoming far too attractive to MaaS providers. Automated triage solutions and sandboxes can help uncover some of these protected samples, but it may not be feasible or cost effective for an organization to run every installation package or installer they utilize through a sandbox.

As this MSI delivery avenue is less and less successful, DarkGate may switch to alternate means of nesting additional stages, but as of writing, other recent samples can be dissected by applying a similar routine to that above.

Being able to triage samples manually when signature-based scanning fails, or reputation checks are bypassed due to the use of a code signing certificate can be crucial when threat hunting, or responding to incidents within an organization that may not have access to a sandbox or automated triage products.

🏂 DarkGate-Loader FileManager :									
2									
WinWorld @ WINWOR	😨 Name	Oreation Date							
	arcldr.exe	07/12/1999 04:00							
Desktop	arcsetup.exe	07/12/1999 04:00							
Downloads	AUTOEXEC.BAT	26/08/2018 14:51							
	boot.ini	26/08/2018 15:38							
Pictures	CONFIG.SYS	26/08/2018 14:51							
Appdata	Documents and Settings	26/08/2018 15:38							
	IO.SYS	26/08/2018 14:51							
	MSDOS.SYS	26/08/2018 14:51							
< >>	NTDETECT.COM	07/12/1999 04:00							

Figure 34: DarkGate File Manager [21]



Figure 35: DarkGate Miscellaneous Features [21]



Figure 36: DarkGate Remote Access Features [21]

References, Appendix, & Tools Used

[1] <u>https://www.sans.edu/cyber-security-programs/bachelors-degree/</u>
[2] <u>https://www.virustotal.com/gui/file/693ff5db0a085db5094bb96cd4c0ce1d1d3fdc2fbf6b92c32836f3</u> <u>e61a089e7a</u>
[3] <u>https://www.globalsign.com/en</u>
[4] <u>https://legroom.net/software/uniextract</u>

[5] <u>https://7-zip.org/</u>

- [6] https://github.com/horsicq/DIE-engine/releases
- [7] https://dmcxblue.gitbook.io/red-team-notes/persistence/dll-search-order-hijacking
- [8] https://hex-rays.com/ida-pro/
- [9] https://ghidra-sre.org/
- [10] https://binary.ninja/
- [11] https://mh-nexus.de/en/hxd/
- [12] https://www.autoitscript.com/site/autoit/
- [13] https://github.com/gchq/CyberChef
- [14] https://github.com/PonyPC/myaut_contrib
- [15] https://www.autoitscript.com/autoit3/docs/functions/StringSplit.htm
- [16] https://www.autoitscript.com/autoit3/docs/functions/FileRead.htm
- [17] https://learn.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-virtualprotect
- [18] https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-enumwindows
- [19] https://www.autoitscript.com/autoit3/docs/functions/BinaryToString.htm
- [20] https://x64dbg.com/
- [21] https://github.security.telekom.com/
- [22] https://malpedia.caad.fkie.fraunhofer.de/details/win.darkgate

Indicators of Compromise

SHA-256 Hashes:

693ff5db0a085db5094bb96cd4c0ce1d1d3fdc2fbf6b92c32836f3e61a089e7a 599ab65935afd40c3bc7f1734cbb8f3c8c7b4b16333b994472f34585ebebe882 107b32c5b789be9893f24d5bfe22633d25b7a3cae80082ef37b30e056869cc5c f049356bb6a8a7cd82a58cdc9e48c492992d91088dda383bd597ff156d8d2929 17158c1a804bbf073d7f0f64a9c974312b3967a43bdc029219ab62545b94e724 2693c9032d5568a44f3e0d834b154d823104905322121328ae0a1600607a2175 237d1bca6e056df5bb16a1216a434634109478f882d3b1d58344c801d184f95d 2296f929340976c680d199ce8e47bd7136d9f4c1f7abc9df79843e094f894236 91274ec3e1678cc1e92c02bc54a24372b19d644c855c96409b2a67a648034ccf ee1ffb1f1903746e98aba2b392979a63a346fa0feab0d0a75477eacc72fc26a6 f7e97b100abe658a0bad506218ff52b5b19adb75a421d7ad91d500c327685d29

C2 Domain, IP & Port: "prodomainnameeforappru[.]com", <u>46.21.157.142:port 443</u>