RagnarLocker Ransomware Threatens to Release Confidential Information

Stories

The latest cybersecurity trends, best practices,
security vulnerabilities, and more

ARCHIVED STORY

Alexandre Mundo · JUN 09, 2020

EXECUTIVE SUMMARY

The RagnarLocker ransomware first appeared in the wild at the end of December 2019 as part of a campaign against compromised networks targeted by its operators.

The ransomware code is small (only 48kb after the protection in its custom packer is removed) and coded in a high programming language (C/C++). Like all ransomware, the **goal of this malware is to encrypt all files that it can and request a ransom for decrypting them.**

RagnarLocker's operators, as we have seen with other bad actors recently, threaten to publish the information they get from compromised machines if ransoms are not paid.

After conducting reconnaissance, the ransomware operators enter the victim's network and, in some pre-deployment stages, steal information before finally dropping the ransomware that will encrypt all files in the victim's machines.

The most notable RagnarLocker attack to date saw this malware deployed in a large company where the malware operators then requested a ransom of close to $11 million USD in return for not leaking information stolen from the company. In this report we will talk about the sample used in this attack.

At the time of writing there are no free decryptors for RagnarLocker.

However, certain McAfee products, including personal antivirus, endpoint, and gateway can protect our customers against the threats that we talk about in this report.

RAGNARLOCKER OVERVIEW

The unpacked malware is a binary file of 32 bits that can be found as an EXE file.

FIGURE 1. INFORMATION ABOUT THE MALWARE

FIGURE 1. INFORMATION ABOUT THE MALWARE

As can be seen in the previous screenshot, this sample was compiled on the 6th of April 2020. The attack mentioned earlier took place some days later, but this sample was prepared for the victim, as we will explain later.

Name	malware.exe
Size	48,460 bytes unpacked (can change between samples), packed can be variable
File-Type	EXE 32 bits (can change between samples)
SHA 256	7af61ce420051640c50b0e73e718dd8c55dddfcb58917a3bead9d3ece2f3e929
SHA 1	60747604d54a18c4e4dc1a2c209e77a793e64dde
Compile time	06-04-2020 (can change between samples)

TECHNICAL DETAILS

As we often see with ransomware, RagnarLocker starts preparing some strings of languages for the CIS countries that are embedded within its own code (in Unicode).

FIGURE 2. THE LANGUAGE STRINGS EMBEDDED INTO THE CODE IN THE STACK

FIGURE 1. INFORMATION ABOUT THE MALWARE

The languages that are hardcoded are:

Georgian
Russian
Ukrainian

Moldavian
Belorussian
Azerbaijani
Turkmen
Kyrgyz
Kazakh
Uzbek
Tajik

After preparing these strings, the malware uses the function “GetLocaleInfoW” to get the LOCALE_SYSTEM_DEFAULT language as a string. Once obtained, it will check the system language with the blacklisted languages and, if any of them match, it will terminate itself with the function “TerminateProcess” and with an error result code of 0x29A (as we have seen before with many different malware samples).

FIGURE 3. CHECK OF THE LANGUAGE AGAINST THE BLACKLIST

FIGURE 3. CHECK OF THE LANGUAGE AGAINST THE BLACKLIST

The check against the LOCALE_SYSTEM_DEFAULT is to prevent a user from installing a language they would not otherwise use as a means of avoiding infection. The check is made against the language selected in Windows. Of course, not everyone in these countries will be using a CIS language in Windows so English is also ok to use. As with other ransomware families, there is no guarantee that infection will be avoided if other languages are selected as the default.

After this the malware will get the name of the infected computer with the function “GetComputerNameW” and the username of whoever is actively using the machine at that time with the function “GetUserNameW”.

FIGURE 3. CHECK OF THE LANGUAGE AGAINST THE BLACKLIST

FIGURE 4. GET THE COMPUTER NAME AND THE USERNAME

After this the malware will read two registry keys:

- HKLM\SOFTWARE\Microsoft\Cryptography and the subkey MachineGuid to get the GUID of the victim machine.
- HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion and the subkey “ProductName” to get the name of the operating system.

For this the malware uses the functions “RegOpenKeyExW”, “RegQueryValueExW” and “RegCloseKey” in the hive HKEY_LOCAL_MACHINE. This hive can be read without admin rights.

FIGURE 5. READ FROM THE REGISTRY THE NAME OF OPERATING SYSTEM AND GUIDE

FIGURE 5. READ FROM THE REGISTRY THE NAME OF OPERATING SYSTEM AND GUIDE

Next, RagnarLocker will prepare the first string in the stack with the function “lstrcpyW” and later will start joining the strings with the function “lstrcatW”.

The sequence is first the GUID of the machine, then the Windows operating system name, the user logged in the machine and, finally, the name of the victim machine.

IGURE 6. GET INFORMATION OF THE USER AND MACHINE AND JOIN ALL STRINGS

FIGURE 6. GET INFORMATION OF THE USER AND MACHINE AND JOIN ALL STRINGS

In the screenshot some values were modified to protect my virtual machine. After getting this information and preparing the string, the malware makes a custom hash with each.

For this, the malware will reserve some memory with “VirtualAlloc” and get the size of the string and compute the hash in a very small loop. After this it will format the hash with the function “wsprintfW” to have it as a Unicode string.

FIGURE 7. MAKE THE CUSTOM HASH AND FORMAT AS UNICODE STRING

FIGURE 7. MAKE THE CUSTOM HASH AND FORMAT AS UNICODE STRING

The hashes are made in the following order:

- Machine name (g. 0xf843256f*)
- Name of the user logged into the machine (e.g. 0x56ef3218*)
- GUID of the infected machine (e.g. 0x78ef216f*)
- Name of the operating system (e.g. 0x91fffe45*)
- Finally, the full string with all strings joined (e.g. 0xe35d68fe*)

*The above values have been changed to protect my machine.

After this it will use the function “wsprintfW”, with the template string “%s-%s-%s-%s-%s”, to format the custom hashes together with hyphens between them, but in this case the hashes are in this order:

- GUID

- Operating System Name
- Name of the logged in user
- Name of the infected machine
- Full string with all other strings joined together

FIGURE 8. CREATE CUSTOM HASH OF THE STRINGS AND FORMAT THE FINAL STRING IN A SPECIAL ORDER<

FIGURE 8. CREATE CUSTOM HASH OF THE STRINGS AND FORMAT THE FINAL STRING IN A SPECIAL ORDER

The malware will get the command line of this launch process and will check if it has more than one argument (the first argument is always in C/C++) with the functions "GetCommandLineW", to get the full command line with arguments if it exists, and "CommandLineToArgvW" to get the arguments if they exist.

If there is more than one argument the malware will avoid the next procedure. To keep the normal flow in the technical details section we will put what happens if only one argument exists. In this case the malware will try to make a Windows Event with the name of the formatted string with all hashes, as explained earlier (in our example case above, 78ef216f-91fffe45-56ef3218-f843256f-e35d68fe).

After trying to create the event the malware will check the last error with the function "GetLastError" and compare with ERROR_ALREADY_EXISTS (0xB7). If the event already exists the malware will check a counter with the value at 0x8000 (32768) and, if is not this value, it will increase the counter by one and try again to make the event, check the last error, and so on, until it can finally make the event, reach the counter value, or it reaches the maximum value in the counter (64233). If the event cannot be created the malware will get the pseudohandle to its own process with the function "GetCurrentProcess" and terminate it with the function "TerminateProcess" with the exit code 0x29A.

FIGURE 9. CREATE EVENT LOOP AFTER CHECKING THERE IS ONLY ONE ARGUMENT IN THE COMMAND LINE

FIGURE 9. CREATE EVENT LOOP AFTER CHECKING THERE IS ONLY ONE ARGUMENT IN THE COMMAND LINE

This is done for several reasons:

- The event is created to avoid relaunching another instance of the malware at the same time.
- The check of the counter is made if another instance of the malware is launched, to wait for the previous one to finish before continuing the process (this avoids some issues with the malware checking for crypted files).

- The check of the argument, as we will explain later, can be used to avoid the event behavior so the malware will always try to encrypt files. It is one of the reasons why a vaccine against this malware is useless if the malware operator executes the malware with an argument as simple as “1”.

After this, the malware will try to access in raw mode all units connected to the victim machine in a physical way, preparing the string “\\.\PHYSICALDRIVE%d”. This string will be formatted with the function “wsprintfW”, starting with the first unit that is 0 to a maximum of 16 in a loop. After the format, the malware will use “CreateFileW” and check that it does not return the error “ERROR_INVALID_HANDLE” (that means the unit cannot be accessed or that it does not exist). If this error is returned it will increase the counter and format the string with the new value of the counter. If it can open the handle to the unit in raw mode it will send two commands using the function “DeviceIoControl”.

The commands are:

- 0x7C0F4 -> IOCTL_DISK_SET_DISK_ATTRIBUTES with the attributes of: DISK_ATTRIBUTE_READ_ONLY and DISK_ATTRIBUTE_OFFLINE.
- 0x70140 -> IOCTL_DISK_UPDATE_PROPERTIES that will be make the drive update its partition table. As the attributes are updated the malware can be accessed in sharing mode on the disk.

FIGURE 10. CONTROL THE PHYSICAL DISK TO HAVE ACCESS TO IT

FIGURE 10. CONTROL THE PHYSICAL DISK TO HAVE ACCESS TO IT

The ransomware’s next action is checking the units that exist and can be accessed without any problem. This can be done in two ways, the first of which is using the functions “FindFirstVolumeA”, “FindNextVolumeA” and “FindVolumeClose”.

FIGURE 11. GET VOLUME LETTER AND INFORMATION TO CHECK IT EXISTS AND CAN BE ACCESSED

FIGURE 11. GET VOLUME LETTER AND INFORMATION TO CHECK IT EXISTS AND CAN BE ACCESSED

The first two functions return the volume and the special internal value associated with it. This information comes from Windows, so the malware needs to translate it to the logic unit letter associated to this volume. This is done with the function “GetVolumePathNamesForVolumeNameA” that will return the logic letter associated to the volume inspected.

With this letter the function “GetVolumeInformationA” is then used to get information of the volume if it exists and is enabled. If the volume does not exist or cannot be checked the function will fail and the volume ignored, and the process will move onto the next volume in the machine.

Another check is made using the function “GetLogicalDrives” that will return a structure and, by checking one byte, the malware will know if the unit exists or not.

After this, the malware will prepare the keys that later will be needed to encrypt the files. To make them it will get the crypto context with the function “CryptAquireContextW” that will generate random data with “CryptGenRandom” and prepare to permutate this value with the SHA-512 algorithm. These values are the key and nonce of the Salsa20 algorithm that will be used later to encrypt files.

FIGURE 12. AQUIRE CRYPTO CONTEXT AND GENERATE SOME DATA AND PREPARE WITH SHA-512D

FIGURE 12. AQUIRE CRYPTO CONTEXT AND GENERATE SOME DATA AND PREPARE WITH SHA-512

The malware continues decrypting some strings using two steps, one in a big function for the first layer and the other that is used later for the second layer and the final string of the service. The services stopped are:

vss
sql
memtas -> associated with MailEnable
mepocs -> associated with MailEnable
Sophos -> associated with Sophos Antivirus
Veeam -> associated with a program to make backups and save in the cloud
Backup -> associated with Asus WebStorage
Pulseway -> associated with remote control software for IT departments
Logme -> associated with remote control software
Logmein -> associated with a remote control software
Conectwise -> associated with a remote control software
Splastop -> associated with a remote control software
Mysql -> associated with a program of databases
Dfs -> associated with the Distribute File System (Microsoft)

Please note: The services list can change between samples.

After decrypting the strings, the malware accesses the SCManager with the function “OpenSCManagerA”. If it does not want access it will ignore all services and continue onto the next step.

If it can open a handle to it, it will get the status of the service with the function “EnumServicesStatusA” and if the service is stopped already will pass to the next one. The malware calls this function two times, firstly to get the correct size needed for this function, with the last error being checked with ¨GetLastError¨ against the value 0xEA (ERROR_MORE_DATA) (that means that the application needs more memory to fill all information than the function gives).

FIGURE 13. OPEN THE SERVICE MANAGER AND ENUMSERVICESTATUSD

FIGURE 13. OPEN THE SERVICE MANAGER AND ENUMSERVICESTATUS

This memory is reserved and the function called again later, in this case to get the real status and, if not stopped, the malware will open the service with the function “OpenServiceA” and query the status of the service with the function “QueryServiceStatusEx”. If the service is not stopped, it will get all dependencies of the service with “EnumDependentServicesA” and finally it will control the service to stop it with the function “ControlService”.

FIGURE 14. OPEN THE SERVICES AND CONTROL THEM

FIGURE 14. OPEN THE SERVICES AND CONTROL THEM

After this, the malware decrypts a list of the processes that it will try terminating if it finds them in the infected machine. For this decryption, the malware uses a string that converts into an integer and uses this integer as a critical value to decrypt the list.

For this task, the malware will create a snapshot of all processes in the system per this blacklist:

sql
mysql
veeam
oracle
ocssd
dbsnmp

synctime
agntsvc
isqlpussvc
xfssvccon
mydesktopqos
ocomm
dbeng50
sqbcoreservice
excel
infopath
msaccess
mspub
onenote
outlook
powerpnt
steam
thebat
thunderbird
visio
wordpad
winword
EduLink2SIMS
bengine
benetns
beserver
pvlsvr

beremote
VxLockdownServer
postgres
fdhost
WSSADMIN
wsstracing
OWSTIMER
dfssvc.exe
swc_service.exe
sophos
SAVAdminService
SavService.exe

Please note: The processes list can change between samples.

After making the snapshot it will enumerate all processes with the functions “Process32FirstW” and “Process32NextW” and for each process found will call the function “WideCharToMultyByte” to get the size needed to convert the name of the process returned in Unicode into Ascii. Later it reserves memory for the name and calls the same function to make the string conversion.

FIGURE 15. GET ALL SYSTEM PROCESSES

FIGURE 15. GET ALL SYSTEM PROCESSES

If the malware, after comparison with the function “StrStrIA”, detects some of the blacklisted processes it will open the process with the function “OpenProcess” and terminate it with the function “TerminateProcess” and with the exit code of 0x29A.

FIGURE 16. OPEN THE PROCESS AND TERMINATE IT IF IT IS BLACKLISTED

FIGURE 16. OPEN THE PROCESS AND TERMINATE IT IF IT IS BLACKLISTED

The malware will check for all processes in the blacklist, using part of the string rather than the exact name. Not using the extension allows for greater obfuscation but carries the risk that some processes could be closed by accident if they share that string.

After this the malware will check if the operating system is 64-bit or not with the function “GetNativeSystemInfo” against the value 9 (that means that the OS is 64-bit).

If the operating system is 64-bit it will get, using “LoadLibraryW” and “GetProcAddress”, the function “Woe64EnableWow64FsRedirection” to remove the redirection that by default is found in 64-bit operating systems. This call is done in a dynamic way, but the malware does not check that the function was retrieved with success; usually it will be, but it is not 100% certain and a crash calling a null pointer could ensue.

FIGURE 17. CHECK THE OPERATING SYSTEM AND DISABLE REDIRECTION IF NEEDED

FIGURE 17. CHECK THE OPERATING SYSTEM AND DISABLE REDIRECTION IF NEEDED

After this, the malware will prepare a string in Unicode embedded in the code with the string “wmic.exe shadowcopy delete” and will call it with the function “CreateProcessW”. After the call it will wait for up to an infinite amount of time using the function “WaitForSingleObject” so that the “wmic.exe” process can finish, irrespective of the size and number of shadow volumes, available machine resources, etc.

Of course, the malware will also use the typical program of “vssadmin” to delete the shadow volumes with the command “vssadmin delete shadows /all /quiet”, as well as with the function “CreateProcessW”. After that it will wait again with “WaitForSingleObject” for the end of the new process.

When it finishes, the malware will check again if the operating system is 64-bit and, if it is, will use “LoadLibraryW” and “GetProcAddress” to get the function “Wo64EnableWow64FsRedirection” to leave the system as before with the redirection. Again, the malware does not check that the function is resolved with success and calls it directly in a dynamic way.

FIGURE 18. DESTROY THE SHADOW VOLUMES AND RE-ENABLE THE REDIRECTION

FIGURE 18. DESTROY THE SHADOW VOLUMES AND RE-ENABLE THE REDIRECTION

While it seems like a mistake to destroy the shadow volumes again, it is not, as RagnarLocker has support for Windows XP and the WMIC classes do not exist in that operating system, hence the need to use the old program “vssadmin” that exists in both new and old operating systems.

The malware continues with the decryption of one PEM block encoded in base64 and the ransom note is prepared for the target in memory.

An example of the ransom note, with confidential information removed, can be seen below:

FIGURE 19. DECRYPTION OF THE PEM BLOCK AND THE RANSOM NOTE

FIGURE 19. DECRYPTION OF THE PEM BLOCK AND THE RANSOM NOTE

An example of the ransom note, with confidential information removed, can be seen below:

FIGURE 20. EXAMPLE REDACTED RANSOM NOTE

FIGURE 20. EXAMPLE REDACTED RANSOM NOTE

After preparing both things the malware decodes the PEM block from the base64 as an object, getting a key that will be used to protect the keys used in the crypto process (of course this procedure may change in future samples as the malware evolves) of the RSA algorithm. It is important to note here that this RSA key changes per sample.

FIGURE 21. DECODE FROM BASE64 AND DECODE THE OBJECT AND IMPORT IT TO USE LATER

FIGURE 21. DECODE FROM BASE64 AND DECODE THE OBJECT AND IMPORT IT TO USE LATER

With this key it will encrypt the two random keys previously generated to protect them in memory. After that, the crypto will release the memory.

Later, it will get the name of the infected machine again, get the size of the name and will calculate the custom hash with the same algorithm as before.

FIGURE 22. CRYPT THE PREVIOUSLY GENERATED VALUES AND GET THE COMPUTER NAME

FIGURE 22. CRYPT THE PREVIOUSLY GENERATED VALUES AND GET THE COMPUTER NAME

With this hash it will prepare a string with this structure:

- RGNR_
- hash from the name of the victim machine
- the extension .txt
- a backslash character at the start of the string

It is done with the function “lstrcatW”.

FIGURE 23. CREATION OF THE RANSOM NOTE NAME

FIGURE 23. CREATION OF THE RANSOM NOTE NAME

With this string it will get the folder of “My Documents” for all users with the function “SHGetSpecialFolderPathW” (this function, based on the operating system, will get different paths for the documents). This string with the path of the folders will join with the string of the ransom note name and later make the final path to create the file.

FIGURE 24. GET THE DOCUMENTS FOLDER TO LATER WRITE THE RANSOM NOTE

FIGURE 24. GET THE DOCUMENTS FOLDER TO LATER WRITE THE RANSOM NOTE

After this it will encode in base64 the critical information to decrypt the files with the function “CryptBinaryToStringA”. The malware uses the function the first time to get the size needed and reserve memory and then uses it again to encode the data. After encoding the data, it creates the ransom note file in the documents path with the string previously joined with the path with the function “CreateFileW” and will write the contents of the ransom note that has been prepared in memory. Later, it will format a special string with some hardcoded characters with “—RAGNAR SECRET—” as a start of block and end of block and, between, will format the encode string in base64 and write in the ransom note.

FIGURE 25. CREATION OF THE RANSOM NOTE AND PUT THE RAGNAR SECRET AT THE END OF IT

FIGURE 25. CREATION OF THE RANSOM NOTE AND PUT THE RAGNAR SECRET AT THE END OF IT

Later, the malware will create a new string with the strings:

- .ragnar_
- hash of the name of the victim machine

This string will be used later as the new extension in the crypted files. After this the malware will enumerate again the logic units of the system with the function “GetLogicalDrivesW” and, to check if the unit is correct, will use the function “GetVolumeInformationW” and check the type of the unit and avoid the type of CD-ROM. For each logic unit it will enumerate all files and folders and will start the crypto process.

FIGURE 26. GET ALL LOGIC UNITS AND CHECK THEM

FIGURE 26. GET ALL LOGIC UNITS AND CHECK THEM

Before starting the crypto process, the malware will try to write the ransom note in the root of each unit that is found as a target.

The malware will ignore folders with these names:

Windows
Windows.old
Internet Explorer
Google
Opera
Opera Software
Mozilla
Mozilla Firefox
$Recycle.Bin
ProgramData
All Users

The ransom note will be written in all folders that are affected and, as with other ransomware, it will use the functions “FindFirstFileW” and “FindNextFileW” to enumerate all contents in each folder.

FIGURE 27. CHECK OF THE BLACKLISTED FOLDER NAMES

FIGURE 27. CHECK OF THE BLACKLISTED FOLDER NAMES

RagnarLocker also avoids crypting certain files:

autorun.inf
boot.ini
bootfont.bin
bootsect.bak
bootmgr
botmgr.efi
bootmgfw.efi
desktop.ini
iconcache.db

ntldr
ntuser.dat
ntuser.dat.log
ntuser.ini
thumbs.db
RGNR_<hash>.txt

FIGURE 28. CHECK OF BLACKLISTED FILE NAMES

FIGURE 28. CHECK OF BLACKLISTED FILE NAMES

If a file has one of these names it will be ignored and, if it has another name, the malware will avoid any file that has these extensions:

.db
.sys
.dll
.lnk
.msi
.drv
.exe

FIGURE 29. CHECK OF BLACKLISTED EXTENSIONS

FIGURE 29. CHECK OF BLACKLISTED EXTENSIONS

These checks are in place to prevent the ransomware from destroying the operating system as the victim needs to have access to the machine to pay the ransom.

For each file that passes all controls a thread will be created that will encrypt it. After creating all threads, the malware will wait for up to an infinite amount of time with the function “WaitForMultipleObjects”.

In the crypto process, in the threads, the malware will check if the file has the mark “_RAGNAR_” at the end with the function “SetFilePointerEx” and by reading 9 bytes and checking if they are this string. If it has this mark the file will be ignored in the crypto process and will be renamed again (with an extension name based on the current machine name).

FIGURE 30. CHECK OF THE MARK OF CRYPTO IN THE FILE

FIGURE 30. CHECK OF THE MARK OF CRYPTO IN THE FILE

In other cases the malware will encrypt the file and at the end of it will write the block crypted of the key, used in a block of 256 bytes, and the nonce used in another block of 256 bytes and, finally, the mark “_RAGNAR_”, along with one byte as NULL to end the string (that makes 9 bytes). The key and nonce used in the Salsa20 algorithm are encrypted by the RSA public key embedded in the malware. This ensures only the malware developers can have the RSA private key that belongs to the public key used to decrypt the key and nonce and, thus, decrypt the files in the system.

Before writing this information, the malware will use the function “LockFile” and, when the process of writing the function is finished, “UnlockFile” to release the file already crypted. This is done to prevent the file being changed or deleted during the encryption process.

FIGURE 31. WRITE THE NEW CONTENTS AT THE END OF THE FILE

FIGURE 31. WRITE THE NEW CONTENTS AT THE END OF THE FILE

FIGURE 32. CHANGE OF THE EXTENSION OF THE CRYPTED FILE

FIGURE 32. CHANGE OF THE EXTENSION OF THE CRYPTED FILE

After all threads of crypto, the malware tries to get the session of the Terminal Services or the session of the user logged in the local machine with the function “WTSGetActiveConsoleSessionId”. With this session it gets the current process of the malware with the function “GetCurrentProcess” and the token of this process with the function “OpenProcessToken”. With the session that was got previously it tries to duplicate the token with the function “DuplicateTokenEx” and sets this token with the function “SetTokenInformation”. After this it will get the system directory with the function “GetSystemDirectoryW” and joins to this path the string “\notepad.exe”.

FIGURE 33. GET THE SESSION OF THE LOCAL USER OR TERMINAL SERVICES AND MANAGE THE TOKENS

FIGURE 33. GET THE SESSION OF THE LOCAL USER OR TERMINAL SERVICES AND MANAGE THE TOKENS

With this prepared, the malware executes Notepad and, as an argument, the ransom note to show to the user what happened in the machine. The function used in this case is “CreateProcessAsUserW” to impersonate the user that had the session previously. Of course, this function is called with the desktop as “WinSta0\Default”.

FIGURE 34. CREATE A PROCESS OF THE NOTEPAD TO SHOW THE RANSOM NOTE

FIGURE 34. CREATE A PROCESS OF THE NOTEPAD TO SHOW THE RANSOM NOTE

After this the malware finishes itself with the function "ExitProcess" and a code of exit of 0.

VACCINE

RagnarLocker can have a vaccine if a program is made that can make the event, as explained in the technical part of this blog. If this event exists, the malware does not make anything in the system, but this type of vaccine is not likely to offer a long-term solution for several reasons:

- The way that the event is done, the malware developers can change the algorithm, or the order of the name of the event, or make a mutex instead of an event and the vaccine will stop working.
- The algorithm has a hardcoded value. If this value is changed the final hash will be different and the vaccine becomes useless.
- The malware is developed in such a way that if it has at least two arguments the event is not created so, if the operators want to execute with safety, they need only to execute with an argument, for example "<malware.exe> 1".
- The malware may evolve over time so the vaccine can be very fragile and limited.

For these reasons we think that a vaccine using this system is not helpful in the longer-term.

CONCLUSION

RagnarLocker is a simple ransomware, much like others that exist in the criminal market. Due to its small size, its operator's aggressive behavior and the knowledge they seem to have that allows them to enter the networks of enterprises, as well as the threat to leak information if the ransom is not paid, RagnarLocker could potentially become a big threat in the future. Time will tell if RagnarLocker becomes a serious threat or disappears against a backdrop of other ransomware with more resources. The code is medium in quality.

COVERAGE

McAfee can protect against this threat in all its products, including personal antivirus, endpoint and gateway.

The names that it can have are:

Ransom-ragnar

Also, learn how Enhanced Remediation, a new capability in ENS 10.7, can automatically rollback changes made by processes that exhibit malicious behavior.

MITRE ATT&CK COVERAGE

- Command and Control : Standard Application Layer Protocol
- Defense Evasion : Disabling Security Tools
- Discovery : Security Software Discovery
- Discovery : Software Discovery
- Discovery : System Information Discovery
- Discovery : System Service Discovery
- Discovery : System Time Discovery
- Discovery : Query registry
- Execution : Command-Line Interface
- Execution : Execution through API
- Exfiltration : Data Encrypted
- Impact : Data Encrypted for Impact
- Impact : Service Stop

YARA RULES

```
rule RagnarLocker
{

/*

This YARA rule detects the ransomware RagnarLocker in memory or unpacked in
disk for the sample with hash SHA1 97f45184770693a91054075f8a45290d4d1fc06f
and perhaps other samples

*/

meta:

author = “McAfee ATR Team”

description = “Rule to detect unpacked sample of RagnarLocker”

version = “1.0”

strings:

$a = { 42 81 F1 3C FF 01 AB 03 F1 8B C6 C1 C0 0D 2B F0 3B D7 }

condition:

$a
```

```
}

import “pe”

rule ragnarlocker_ransomware

{

meta:

description = “Rule to detect RagnarLocker samples”

author = “Christiaan Beek | Marc Rivero | McAfee ATR Team”

reference = “https://www.bleepingcomputer.com/news/security/ragnar-locker-
ransomware-targets-msp-enterprise-support-tools/”

date = “2020-04-15”

hash1 =
“63096f288f49b25d50f4aea52dc1fc00871b3927fa2a81fa0b0d752b261a3059”

hash2 =
“9bdd7f965d1c67396afb0a84c78b4d12118ff377db7efdca4a1340933120f376”

hash3 =
“ec35c76ad2c8192f09c02eca1f263b406163470ca8438d054db7adcf5bfc0597”

hash4 =
“9706a97ffa43a0258571def8912dc2b8bf1ee207676052ad1b9c16ca9953fc2c”

strings:

//—RAGNAR SECRET—

$s1 = {2D 2D 2D 52 41 47 4E 41 52 20 53 45 43 52 45 54 2D 2D 2D}

$s2 = { 66 ?? ?? ?? ?? ?? ?? 66 ?? ?? ?? B8 ?? ?? ?? ?? 0F 44 }

$s3 = { 5? 8B ?? 5? 5? 8B ?? ?? 8B ?? 85 ?? 0F 84 }

$s4 = { FF 1? ?? ?? ?? ?? 3D ?? ?? ?? ?? 0F 85 }

$s5 = { 8D ?? ?? ?? ?? ?? 5? FF 7? ?? E8 ?? ?? ?? ?? 85 ?? 0F 85 }

$op1 = { 0f 11 85 70 ff ff ff 8b b5 74 ff ff ff 0f 10 41 }

$p0 = { 72 eb fe ff 55 8b ec 81 ec 00 01 00 00 53 56 57 }

$p1 = { 60 be 00 00 41 00 8d be 00 10 ff ff 57 eb 0b 90 }

$bp0 = { e8 b7 d2 ff ff ff b6 84 }

$bp1 = { c7 85 7c ff ff ff 24 d2 00 00 8b 8d 7c ff ff ff }
```

```
$bp2 = { 8d 85 7c ff ff ff 89 85 64 ff ff ff 8d 4d 84 89 }
condition:
uint16(0) == 0x5a4d and
filesize < 100KB and
(4 of ($s*) and $op1) or
all of ($p*) and pe.imphash() == “9f611945f0fe0109fe728f39aad47024” or
all of ($bp*) and pe.imphash() == “489a2424d7a14a26bfcfb006de3cd226”
}
```

IOCs

SHA256	7af61ce420051640c50b0e73e718dd8c55dddfcb58917a3bead9d3ece2f3e929
SHA256	c2bd70495630ed8279de0713a010e5e55f3da29323b59ef71401b12942ba52f6
SHA256	dd5d4cf9422b6e4514d49a3ec542cffb682be8a24079010cda689afbb44ac0f4
SHA256	63096f288f49b25d50f4aea52dc1fc00871b3927fa2a81fa0b0d752b261a3059
SHA256	b670441066ff868d06c682e5167b9dbc85b5323f3acfbbc044cabc0e5a594186
SHA256	68eb2d2d7866775d6bf106a914281491d23769a9eda88fc078328150b8432bb3
SHA256	1bf68d3d1b89e4f225c947442dc71a4793a3100465c95ae85ce6f7d987100ee1
SHA256	30dcc7a8ae98e52ee5547379048ca1fc90925e09a2a81c055021ba225c1d064c

USING MVISION EDR TO DETECT RAGNARLOCKER

With thanks to Mo Cashman and Filippo Sitzia

We downloaded a RagnarLocker sample from Virus Total to test detection capability by MVISION Endpoint Detection and Response (EDR). We tested first with the original sample which was known to most detection engines by this time. We then changed file hashes to test detection with an unknown sample. In both cases, MVISION EDR identified the suspicious behaviors and raised alerts. The original sample was detected as a HIGH Risk because the file had a known malicious reputation in McAfee Global Threat Intelligence which is integrated with MVISION EDR. The unknown samples were detected as Medium Risk and most likely would have triggered further inspection by a security analyst.

Sample VT submission

2020-05-30 13:30:55, File size: 48.50 KB, File type Win32 EXE, File name: omniga.exe, VT detections: 51/73

Test Environment

OS Win10, ENS 10.7 Threat Protection off, Adaptive Threat Protection off, MVISION EDR

Execution with original HASH – 3bc8ce79ee7043c9ad70698e3fc2013806244dc5112c8c8d465e96757b57b1e1

To further test MVISION EDR effectiveness, we modified the hash file slightly:

Execution with HASH changed – 63F5B6ED99C559341CF1AD081BAF55B4EACAD8E46D056764531BD316BF3C3EE3

Alerting Results for both samples