Taking a deep dive into Sodinokibi ransomware

By Acronis

Archived: 2026-04-05 18:58:32 UTC

While Sodinokibi ransomware has been in the news recently, technical details for that particular strain have been far less visible. In this article, we'll dissect Sodinokibi, shine a light on how it works, and review how you can protect your system from this threat.

Researched and written by Ravikant Tiwari and Alexander Koshelev

Executive Summary

- Sodinokibi is likely being distributed by attackers affiliated with those that distributed the infamous GandCrab ransomware family, which is supposed to be retired soon according to the underground forum where GandCrab first appeared.
- Sodinokibi ransomware exploits an Oracle WebLogic vulnerability (CVE-2019-2725) to gain access to the victim's machine. Once it's in, the malware tries to execute itself with elevated user rights in order to access all files and resources on the system without any restriction.
- Sodinokibi tries to avoid infecting computers from Iran, Russia, and other countries that were formerly part of the USSR.
- This ransomware strain uses AES and Salsa20 algorithms to encrypt user's files, AES is used to encrypt session keys and data that is sent to the control server, user files are encrypted using Salsa20 encryption.
- Sodinokibi uses an Elliptic-curve Diffie-Hellman key exchange algorithm to generate and propagate encryption keys.
- Once it infiltrates a machine, it wipes out all of the files in the backup folder.
- Currently, the ransomware demands 0.32806964 BTC (≈ $2,500) to regain access to the encrypted files. They claim that this amount should be paid within four days or the ransom demand will be doubled.

Sodinokibi ransomware

How it works

The Sodinokibi ransomware sample we analyzed was packed using a custom packer. Even after successful unpacking, the main Sodinokibi code does not seem to have much of a readable string. Neither does it have any imports for system libraries and APIs, which means a static AV scanner that depends on a readable string and imported API table will have a hard time detecting it.

API names and other required strings are decrypted during its runtime using the RC4 algorithm. To make the situation even more challenging for anti-virus software, most of the operations on strings are performed using a DJB hash of the string rather than the string itself.

Initialization

Sodinokibi starts by building a dynamic import table and ensuring that this is the only instance running currently on the system with the help of mutexes. Once the mutex check is passed, it decrypts the JSON config stored within the binary using RC4 and checks for the Boolean key value “exp”. If it is set to “true” Sodinokibi tries to run an exploit. In our case, the value of “exp” key is set to true so it proceeds to run the exploitation function.

Sodinokibi - Decrypted JSON Configuration

Decrypted JSON Configuration

Figure 1: Decrypted JSON Configuration

The code responsible for running the exploit first checks if the September 11, 2018 patch (KB4457138) is applied on the machine. This patch addresses multiple vulnerabilities mentioned below. If the patch is not detected, the ransomware proceeds to execute a 32- or 64-bit version of the shellcode depending on the platform architecture. We believe it tries to elevate its privilege by exploiting CVE-2018-8440.

Figure 2: Snippet 1

Vulnerabilities addressed by patch KB4457138 are listed in the table below:

If the system is not vulnerable and the process is still running as a limited user, it will use a RUNAS command to launch another instance with administrative rights and terminate the current instance if it is running with limited privileges. The complete flow can be seen in the code below.

Sodinokibi ransomware

Figure 3

After Sodinokibi successfully starts in Admin mode, it does an extra pre-check based on “bro” key in the JSON configuration and country. It will try not to infect computers from the following countries based on the locale setting of the computer.

Figure 4: Matching language IDs

Figure 4: Matching language IDs

Exempted Countries list

Romania

Russia

Ukraine

Belarus

Estonia

Latvia

Lithuanian

Tajikistan

Iran

Armenia

Azerbaijan

Georgia

Kazakhstan

Kyrgyzstan

Turkmenistan

Uzbekistan

Tatarstan

After passing the pre-check it terminates the mysql.exe process (if it's running) so that it can gain access to MySQL files for encryption, then deletes all Windows SHADOW COPIES (Windows built-in backup mechanism) using vssadmin, and disables Windows recovery using bcdedit (boot policy editor) as shown below:

vssadmin.exe Delete Shadows /All /Quiet & bcedit /set {default} recoveryenabled No & bcedit /set {default} bootstatuspolice ignorealfailures

Figure 5

Before encrypting user files, Sodinokibi searches the entire file system and network shares for all directories named "backup" and wipes it out. Interestingly, before wiping all the files inside this directory it overwrites the content with random bytes to make file recovery impossible. Fortunately, Acronis Backup files can't be deleted easily, as they are protected using kernel mode drivers to thwart such illicit deletion by ransomware.

Key Generation

Sodinokibi uses an Elliptic-curve Diffie–Hellman (ECDH) key generation and exchange algorithm to generate a private-public key pair. It uses this to generate a shared secret, which will be used as the key for symmetric encryption algorithms AES and Salsa20 which are used to encrypt different kinds of data.

- AES encryption is used to encrypt the private keys of the private-public key pair, which is generated locally on the user's machine as well as the data sent over the network.
- Salsa20, on the other hand, is used for encrypting user files.

Sodinokibi is shipped with two different public keys, one as part of JSON configuration and another embedded in the binary itself. These public keys will be used to encrypt the locally-generated private key. Below we detail the steps included in the key generation and encryption process.

Step 1.Generate a session private (secret, random number) and public key pair on the local machine.

Generating local public and private keys

Figure 6: Generating local public and private keys

Encrypting the private key from Step 1 using the public key present in JSON configuration

Step 2. Generate another private and public key pair.

Step 3. Use the private key from Step 2 and the public key (pk key value) from JSON to generate a shared key and hash it to generate a symmetric key.

Generating a symmetric key using a shared key

Figure 7: Generating a symmetric key using a shared key

Step 4. Generate a 16-byte IV (initialization vector).

Step 5. Encrypt the private key generated in Step 1 using AES encryption with the Key and IV generated in Steps 3 and 4.

Step 6. Calculate CRC32 of the encrypted private key generated in Step 5.

Step 7. Append IV and CRC32 at the end of the buffer containing the encrypted private key from Step 5.

Step 8. Save this buffer to a mapped file offset marked as “sk_key” in memory.

Figure 8: Encrypting the private key from step one using the attacker's public keys

Figure 8: Encrypting the private key from step one using the attacker's public keys

Encrypting the private key from Step 1, using public key present embedded in the binary

Step 9. Repeat Steps 2 through 7 by using a different public key that comes embedded in the binary for Step 3.

Step 10. Save this buffer to a mapped file offset marked as “0_key” in the memory

The sk_key, 0_key, and pk_key are written to the following registry key respectively depending on access rights.

HKLM\SOFTWARE\recfg\sk_key OR HKCU\SOFTWARE\recfg\sk_key

HKLM\SOFTWARE\recfg\0_key OR HKCU\SOFTWARE\recfg\0_key

HKLM\SOFTWARE\recfg\pk_key OR HKCU\SOFTWARE\recfg\pk_key

Encrypted secret key in registry

Figure 9: Encrypted secret key in registry

Generating per file keys for Salsa20

Step 11. Generate a new private and public key pair.

Step 12. Generate a shared key using the session public key generated in Step 2 and hash it to get another symmetric key for using in Salsa20 key generation.

Step 13. Set up a 256-bit (32 bytes) Salsa20 key state

Step 14. Generate an 8-bit IV for Salsa20 key

Step 15. Generate a Salsa20 key

Step 16. Use this Salsa20 key_state for encrypting user files using Salsa20 encryption.

Generation of per file Salsa20 key

Figure 10: Generation of per file Salsa20 key

Repeat Steps 11 to 16 for each file that is to be encrypted.

Encryption and decryption illustration

To understand how the keys are generated and transferred between attacker and the victim’s machine we need to take a look at how Diffie Hellman works with the help of the image below.

Figure 11: Elliptic-Curve Diffie-Hellman (ECDH) Key Exchange

Sodinokibi encryption process in detail

Figure 12: Encryption Process

Sodinokibi decryption process in detail

The process of decryption will require the knowledge of Attacker’s private key which is not publicly disclosed and thus it’s not possible to restore the files back.

Figure 13: Decryption Process (Attacker’s seceret is Attacker’s Private key)

The simplified version of how the decryption process of the user files will look like the graph below.

Figure 14

File Encryption on Local Hard Drives and Network Share

To encrypt user files Sodinokibi uses I/O Completion Ports and creates multiple encryption threads to a maximum of twice the number of processor cores present on the machine and associate these threads to the created I/O

completion port. These threads use GetQueuedCompletionStatus Win API function to wait for a completion packet to be queued to the I/O completion port before they proceed to the file encryption.

Once the threads are created and waiting for I/O packets to arrive, Sodinokibi starts enumerating user files on all the local drives and network shares except CDROM and RAMDISK drives and begins associating files which are not in the exempted folder, file or file extension list to this I/O completion port by calling AddFileToIoCompletionPort user function and calls PostQueuedCompletionStatus Win API to post an I/O packet on the I/O completion ports which will trigger the thread waiting on this I/O completion port to resume and proceed to encrypt files. AddFileToIoCompletionPort also generates a unique Salsa20 key for each file that is to be encrypted and pass this Salsa20 key to the encrypting thread as part of structure containing other metadata that has to be written to file as well after encryption using lpOverlapped parameter of PostQueuedCompletionStatus Win API. During enumeration it also create a ransom note file in all folders that are not in the exempted folder list. Once there are no more files to enumerate the main thread waits in a loop until total number of files encrypted and renamed equals to the total number of files added to the I/O completion port for encryption.

Finally, it sets a flag which indicates there are no more files to enumerate and posts multiple I/O completion packets, by doing this it makes sure that extra threads waiting for files should resume and break the execution flow to finish immediately.

Figure 15

Exempted Folders

- "$windows.~bt"
- "intel"
- "program files (x86)"
- "program files"
- "msocache"
- "$recycle.bin"
- "$windows.~ws"
- "tor browser"
- "boot"
- "system volume information"
- "perflogs"
- "google"
- "application data"
- "windows"
- "programdata"
- "windows.old"
- "appdata"
- "mozilla"

Exempted Files

- "bootfont.bin"

- "boot.ini"
- "ntuser.dat"
- "desktop.ini"
- "iconcache.db"
- "ntldr"
- "ntuser.dat.log"
- "thumbs.db"
- "bootsect.bak"
- "ntuser.ini"
- "autorun.inf"

Exempted File extensions

- "themepack"
- "ldf"
- "scr"
- "icl"
- "386"
- "cmd"
- "ani"
- "adv"
- "theme"
- "msi"
- "rtp"
- "diagcfg"
- "msstyles"
- "bin"
- "hlp"
- "shs"
- "drv"
- "wpx"
- "deskthemepack"
- "bat"
- "rom"
- "msc"
- "lnk"
- "cab"
- "spl"
- "ps1"
- "msu"
- "ics"
- "key"
- "msp"

- "com"
- "sys"
- "diagpkg"
- "nls"
- "diagcab"
- "ico"
- "lock"
- "ocx"
- "mpa"
- "cur"
- "cpl"
- "mod"
- "hta"
- "exe"
- "icns"
- "prf"
- "dll"
- "nomedia"
- "idx"

The encrypting thread takes care of reading the file contents, encrypting it, writing it back to the same file, writing metadata that contains encrypted session Private key the per file ECDH Public key and per file Salsa20 IV used for encrypting the files and then renaming it by appending with a randomly generated extension name. File are encrypted using Salsa20 (Chacha variant) encryption algorithm inside EncryptAndWrite user function.

The snippet below shows the code for EncryptingThreadRoutine user function.

Figure 16

File Structure after Encryption

Figure 17: Structure of encrypted file

Network Activity

After the encryption process is complete, the ransomware prepares the data to send to the control server. This data contains different fields from the JSON configuration, system information, and encryption keys. Prepared data is also saved to the registry key “[HKLM|HKCU]\SOFTWARE\recfg\stat” before encrypting it with AES and sending it to the attacker’s server.

Figure 18: Data sent over network

Figure 18: Data sent over network

pid

Value of ‘*pid’* key from JSON configuration file.

sub

Value of ‘*sub’* key in JSON configuration file.

uid

crc32(Volume serial number) + crc32(Processor Model)

grp

Network domain name of the system

lng

Language of the system

bro

True or false based on exempted country

bit

Platform architecture (32|64 bit)

The information is sent to randomly generated URL which is in the form

where:

Domain name

sochi-okna23[.]ru

Path part 1

"wp-content" "static" "content" "include" "uploads" "news" "data" "admin"

Path part 2

"images" "pictures" "image" "temp" "tmp" "graphic" "assets" "pics" "game"

Generating URL

Figure 19: Generating URL

Ransom Note

Sodinokibi contains a template of its ransom note with placeholders for user-specific details. These placeholders are dynamically substituted with user-specific extension name, user id (uid – see the table above for description), and key. The ransom note is placed in each directory excluding the whitelisted one.

Figure 20

Decryption

There is no free decrypter available for this ransomware and the only choice is to use the decryption service provided by the attackers, which can be accessed by following the instructions in the ransom note.

The decryption is below

Figure 21

Conclusion

To protect against ransomware we recommend using an advanced anti-ransomware solution and maintain an updated anti-virus solution. All Acronis product are equipped with advanced anti-ransomware technology and can protect you against any such attack and minimize the risk of data loss.

Cyber protecton products like the personal solution Acronis True Image 2020 or business solution Acronis Backup come with the AI-based anti-malware defense Acronis Active Protection built in, and are therefore able to protect users against Sodinokibi ransomware.

Figure 22: Acronis Active Protection detects Sodinokibi
