Ransomware Encryption Goes Wrong

By Aaron Gdanski, Limor Kessem

Published: 2021-11-01 · Archived: 2026-04-05 21:15:18 UTC

Author

Limor Kessem

X-Force Cyber Crisis Management Global Lead

IBM

IBM Security X-Force researchers have recently reverse-engineered Prometheus ransomware samples as part of ongoing incident response operations. X-Force has found that samples that infected organizational networks featured flawed encryption. This allowed our team to develop a fast-acting decryptor and help customers recover from the attack without a decryption key.

While rare, ransomware developers can make mistakes in the ways they implement encryption, causing unintended flaws. This is not the first time X-Force sees faulty encryption mechanisms save the day for victimized organizations. Mistakes can easily occur when malware developers use patchwork code and dabble in cryptography without appropriate expertise.

Most organized cybercrime groups do use properly configured encryption, which is almost always impossible to break. That said, the option to examine possibilities can make a difference for victimized organizations and change the course of negotiation and recovery.

In early 2020, a new ransomware family dubbed "Thanos" was discovered on sale in underground forums mostly frequented by cybercriminals. At the time, Thanos was advertised as a "Ransomware Affiliate Program," available for anyone to buy. The malware saw regular updates and new features added over time. A closer look at its code revealed that it was also used at the baseline in ransomware samples that were tracked as "Hakbit" and used in additional attacks that targeted organizations in Austria, Switzerland and Germany.

Thanos' developer equipped it with a bootlocker in mid-2020 and was also using a somewhat novel technique of encrypting files known as "RIPlace," in which they weaponized research into ransomware evasion techniques based on file characteristics.

In September 2020, Thanos was detected in attacks on government organizations in MEA. It presented the victims with a black screen that demanded money to unlock files, and while it had a supposed capability to run a destructive attack, that function did not work and left MBR intact.

By June 2021, more of Thanos made headlines, only this time as the base code for another ransomware, Prometheus. The latter was used in double-extortion attacks that encrypted files but also stole data and threatened to release it unless a hefty ransom was paid. Prometheus' operators claimed to be part of the REvil group, they

even placed a logo of sorts on their demands for ransom but provided no proof to that effect and may have wanted to use that as a pressure tactic.

While the original Thanos is not as active, its code does not rest. In mid-2021 it was detected in further ransomware attacks, this time used by a group going by the name "Haron."

The Thanos code itself was and is being used by multiple threat actors, some of which were suspected to have nation-state sponsored ties. The Prometheus variant has died out in recent months, but other variations can continue to rise from the same Thanos base. What changes through each variation is customization. In Prometheus' case, its operators used social engineering well, but were not as adept at working with encryption.

While working on Prometheus samples that encrypted files on infected devices, IBM Security X-Force researchers uncovered a weakness in the key generation algorithm used in the encryption process. Unlike most ransomware cases, this was good news that ended up helping a victimized organization.

Our analysis showed that to generate the seed for encryption, the algorithm Prometheus selected uses a hardcoded initialization vector (IV) and the uptime of the computer. This means that the seed value is a lot easier to guess than it should be, since certain parameters about the encrypted file and the infected device can be obtained.

Based on such parameters, X-Force wrote a decryptor that ended up working quickly to decrypt file types that had known file headers, for example: pdf, doc, xls, ppt, docx, xlsx, pptx, 7z, mp3, jpg, jpeg, zip, iso, exe, dll, sys and png. Decrypting the files was made even easier when device boot time was known. Boot times are not a parameter one would have to guess, they can be obtained via the CBS.log file in the Windows directory.

Using the decryptor was a great option for the recovery process X-Force supported, but another note is important here. Some open-source decryption tools may emerge over time and might seem like a recovery tool that can help in large-scale cases. One must consider the time it takes a decryptor to unlock each file. Some open-source tools can take around five hours per file, or more, which would be too time consuming in cases where a lot of data is no longer accessible. A reasonable amount of time to decrypt each file should be a few minutes or less.

In the Prometheus variants analyzed, there are two ways the ransomware can be configured for encryption:

Configuration 1

Encryption process per file:

- A 32-byte string is generated using C#'s Random class. The default constructor is used, which passes Environment.TickCount as the seed.
- The string is then encrypted using a hard-coded RSA public key. PKCS#1 v1.5 padding is used. The ciphertext is then Base64 encoded.

- The file is encrypted using a symmetric algorithm (Salsa20) with a hardcoded 8-byte array as the initialization vector (IV).
- The key is the 32-byte string described above. The ciphertext is written to the encrypted file.
- The encrypted, Base64 encoded key is then appended to the end of the encrypted file, along with the string 'GotAllDone'.

Configuration 2

Encryption process per file:

- A 32-byte string is generated using C#'s Random class. The default constructor is used, which passes Environment.TickCount as the seed.
- The string is then encrypted using a hard-coded RSA public key. PKCS#1 v1.5 padding is used. The ciphertext is then Base64 encoded.
- RFC2898DeriveBytes is used to generate a 32-byte key and an 8 byte IV. The Rfc2898DeriveBytes Class implements password-based key derivation functionality, PBKDF2, by using a pseudo-random number generator. The string generated above is used as the password, and the salt is a hardcoded 8-byte array.
- The file is encrypted using a symmetric algorithm using the parameters generated above. The ciphertext is written to the encrypted file.
- The encrypted, Base64 encoded key is then appended to the end of the encrypted file, along with the string 'GotAllDone'.

Weaknesses in this encryption methodology

X-Force found this technique to be lacking in a way that allowed for finding a way to decrypt affected files.

C#'s Random class will generate the exact same bytes as long as the seed is known. In this case, the seed is the Environment.TickCount variable, which is the number of milliseconds elapsed since a computer was last started.

That seed value can be guessed given certain parameters. Moreover, the Environment.TickCount variable is also updated around every 16 milliseconds, so it is possible for multiple files to have the same key, which can make decryption even faster down the line.

The hardcoded IV provided no additional security in this case, considering it can easily be obtained and appears to be the same for every sample analyzed. To make encryption stronger, the IV should typically be random or pseudorandom.

Can all Prometheus samples be broken in the same way? X-Force's analysis indicates that any Prometheus sample that uses the C# Random class to generate keys is vulnerable. Of note, they only decrypted files that were encrypted using a Salsa20 stream cipher. Some Prometheus ransomware samples can be configured to use AES-256 and while these samples are still vulnerable, X-Force did not test the decryptor on such in their current work.

Requirements

To decrypt files, we would need the following information:

- The hardcoded IV used in the sample. In all samples observed, the IV was an 8-byte array: 1, 2, 3, 4, 5, 6, 7, 8.
- Text or bytes to search for from the decrypted file. X-Force used known file header bytes associated with common file extensions. For example, if the encrypted file's original extension is .pdf, the text to search for in the file to determine success is "%PDF".

Optional data to use

- The configuration of the sample: it's better to determine which configuration is being used for the encryption. For instance, should RFC2898DeriveBytes be used to obtain the key?
- Mtime: the file's modification time as recorded by the infected device.
- The boot time of the infected device. This can be found in the CBS.log file, which is in the Windows directory. This parameter may not have to be exact. Note that Prometheus will not encrypt this file. This file also contains all boot times, meaning it is possible to obtain the boot time even if encryption happened months ago. If the boot time and file modification time are not provided, decryption is still possible but will take significantly longer.

File type limitation

Currently, only files with known file headers can be decrypted. For example: pdf, doc, xls, ppt, docx, xlsx, pptx, 7z, mp3, jpg, jpeg, zip, iso, exe, dll, sys and png.

The following process is what X-Force used in their current work to decrypt data encrypted by Prometheus. It focuses on the malware's first configuration.

Decryption in configuration 1 mode

- Extract the encrypted text from the file intended for decryption. This can be done by removing the junk appended to the end of the file. The amount of junk is equivalent to BASE64_ENCODED_SIZE(RSA_KEY_SIZE) + 'GotAllDone'.Length.
- Attempt to estimate the tick count at the time of encryption. The tick count begins at zero on system boot, is incremented every millisecond, but only updated every 10-15 milliseconds. Tick count continuously loops after hitting INT_MAX.
- Continuously attempt to generate a key and decrypt the ciphertext using the potential seed.
- Decrement the potential seed if the plaintext is not correct. In most cases, the estimated seed should be within 6000 values of the correct seed. It appears that each file should take around 10 seconds to decrypt, without any optimization considered.

Note that during any decryption effort, whether custom-built or provided by ransomware actors, certain conditions can affect the accuracy of time estimates of the decryption. If a file takes longer than desired to unlock, it is likely that any other file from that same device will take a similar amount of time.

Decrypting multiple files from the same infected device

If the seed value is found for the first file encrypted, that seed value can be continuously incremented in order to find the values for every other file. This may provide a slightly faster decryption process for computers with hundreds or thousands of files to decrypt.

The decryptor tool can be run against an entire directory of files or on a per file basis.

The ransomware problem has turned into a true pandemic for organizations. Every month new attacks are detected, and new malware families and variations arise in the commercial cybercrime arena and through closed groups. Companies are struggling to prevent ransomware infections on the one hand and prepare for incidents on the other. Paying cybercriminals has also turned into a high-stake negotiation where the leverage is almost always on the attacker's side.

Will it ever end? With this crime being so rampant in industrialized countries, governments and law enforcement agencies are becoming increasingly involved in ransomware cases, especially in cases where multiple companies are hit.

Stopping attacks is hard because it only takes a small security gap for attackers to find a way in. Response goes a longer way in detecting, containing and helping organizations recover from ransomware attacks. IBM Security X-Force can help. For any other assistance by IBM's team of experts, explore their incident response and threat intelligence services here.

Source: https://securityintelligence.com/posts/ransomware-encryption-goes-wrong/