Shining a light on SunCrypt's curious file encryption mechanism

tesorion.nl/en/posts/shining-a-light-on-suncrypts-curious-file-encryption-mechanism/

By Gijs Rijnders October 8, 2020

SunCrypt is a ransomware family that was first discovered in late 2019 but is currently in the spotlights as the operators claim to be working with the infamous Maze cartel. A detailed analysis by Acronis reported that they also share some attacking techniques.

In this blog post we dive further into the details of SunCrypt's file encryption mechanism and discuss some unusual choices the authors made.

File Encryption

SunCrypt uses a combination of symmetric and asymmetric cryptography which is common in ransomware. The ChaCha20 stream cipher is used for the symmetric part: encrypting the victim's files. Chacha20 supports both 128- and 256-bits keys, in this case the 256-bit variant is used. A new symmetric key is generated at random for each file the

ransomware encrypts. The function RtlGenRandom, or SystemFunction036 is used to generate these keys. The nonce used by the ransomware is zero, but from a security perspective, this is not a problem here.

Acronis reported that the asymmetric encryption is done using RSA in two layers: a randomly generated session key used to encrypt symmetric keys and an embedded public key to encrypt the session key. Upon closer inspection however, we concluded that SunCrypt uses the Curve25519 elliptic curve algorithm, and that it does not use session keys.

The asymmetric part of the encryption scheme is based on the so called: Elliptic-Curve Integrated Encryption Scheme, or ECIES. A public value is computed from the symmetric key using the Curve25519 algorithm. This public value, often referred to as 'R', is stored with the encrypted file. Combined with the master private key from the attackers, this public value can be used to recover the correct key for the file decryption. For more details on the ECIES scheme.

As the ransomware does not use session keys, the symmetric keys are encrypted using the embedded public key. This public key is hardcoded. The screenshot below shows part of the SunCrypt code that performs the ECIES operations for a file, stores the R value discussed above as file extension and initializes the symmetric encryption algorithm. To avoid leaking the key, it is erased in memory directly afterwards.

```
curve25519_compute(symm_r_value, random_file_key, curve25519_basepoint);
encrypted_file_extension = &dot_prefixed_encrypted_file_extension[1];
for ( i = 0; i < 32; ++i )
{
  wsprintfW(encrypted_file_extension, L"%02X", symm_r_value[i]);// Convert R-value byte to hexadecimal
  encrypted_file_extension += 2;
}
qmemcpy(&struc->EncryptedFileExtension, dot_prefixed_encrypted_file_extension, 0x84u);
curve25519_compute(key, random_file_key, &ec_public_key);
chacha20_init_context(&struc->ChaCha20Context, key, chacha20_zero_nonce, 20);
*(_OWORD *)key = 0i64;                  // Erases symmetric key in memory after ChaCha initialization
*(_OWORD *)&key[16] = 0i64;
```

SunCrypt encrypts a single block in every file, starting at the beginning. If the file is smaller than 512 bytes, it is left untouched. If the file is smaller than 32 kilobytes, the entire file is encrypted and otherwise, the first 32 kilobytes are encrypted.

Ransomware commonly stores a block of information at the end of an encrypted file, called a footer. This footer usually contains information for decryption, such as the original file name or size, and checksums for validating the decrypted content.

A peculiar characteristic of SunCrypt is that encrypted files do not have such a footer, while the R value must be stored to facilitate decryption. The ransomware chooses to store this value in the file extension of encrypted files instead. Therefore, every file has a different, 64-character hexadecimal file extension. We have seen this referred to as a 'hash' sometimes, but in the case of SunCrypt it is not a hash, but rather the 'encrypted' key for the file. Instead of checking the footer of a file that is already encrypted by SunCrypt, the ransomware checks whether the extension of a file on the victim's system is

exactly 64 characters long, and if so, leaves the file untouched. Moreover, the footer-less approach also implies that the file is rendered useless if its extension is changed. As no additional information is stored with a file, we also found that SunCrypt does not provide integrity checks for encrypted files. Even though this is not required for successful decryption, corruption cannot be reliably detected.

Damage Control

Tesorion regularly contributes ransomware decryption tools to NoMoreRansom. However, decryption of files is not always possible. Therefore, it is important to implement basic measures for securing your network. Think about offline backups, penetration testing and automated detect & respond. However, these are just some examples of basic measures, each organization will demand specific measures. Would you like more information about possible measures that would fit your organization? Please feel free to contact us! We are happy to help you improve security within your organization in a way that fits your organization best.

Indicators of Compromise

• SHA256 of PowerShell loader:
3090bff3d16b0b150444c3bfb196229ba0ab0b6b826fa306803de0192beddb80
• SHA256 of SunCrypt PE file:
e3dea10844aebc7d60ae330f2730b7ed9d18b5eec02ef9fd4a394660e82e2219