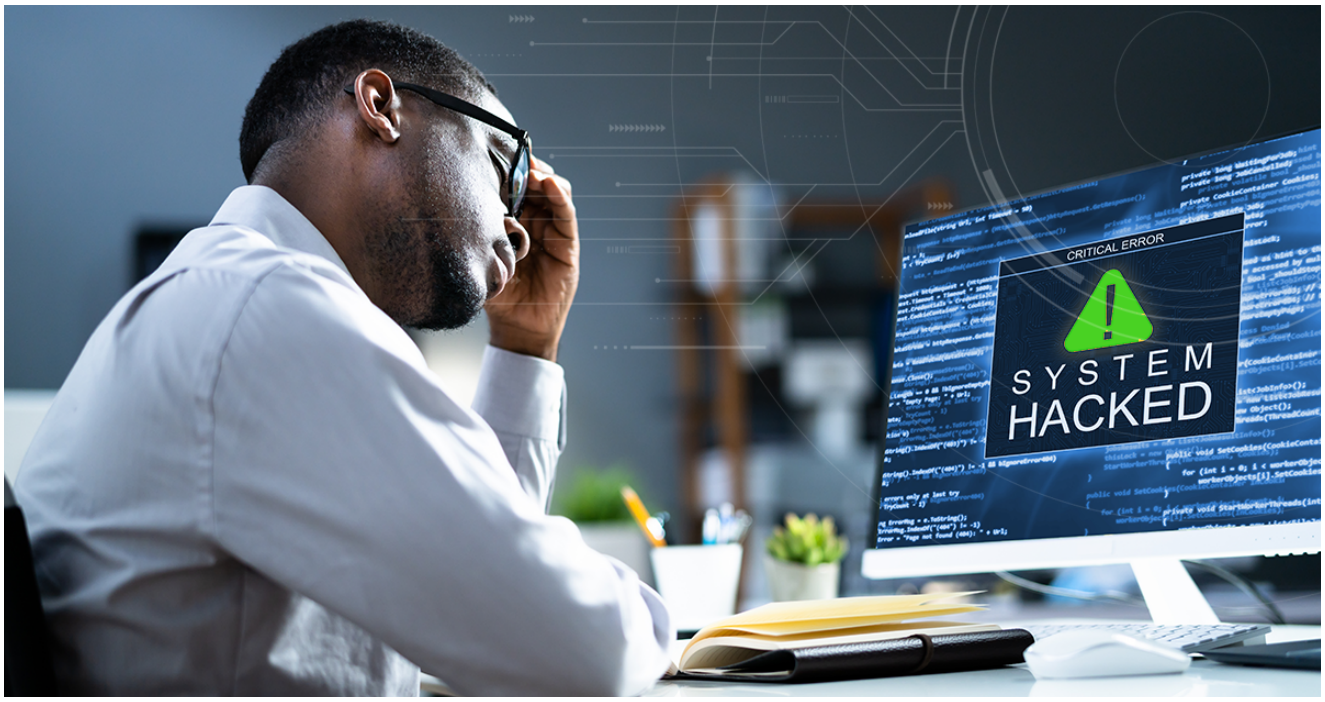
Lorenz ransomware rebound: corruption and irrecoverable files

tesorion.nl/en/posts/lorenz-ransomware-rebound-corruption-and-irrecoverable-files/

By Gijs Rijnders March 21, 2022



In July 2021, we came across the Lorenz ransomware. Lorenz is a ransomware strain that targets organizations, demanding hundreds of thousands of dollars in ransom. In our analysis in 2021 we found that the ransomware contained bugs and that it was possible to build a decryptor. As a result, we published a free decryptor for Lorenz via the NoMoreRansom initiative.

In early March 2022 we came across a new variant of the Lorenz ransomware. The sample we analyzed dates back to March 2, 2022. Files encrypted by this variant are different from the previous one. This blog contains our findings on the new variant. Furthermore, we explain a serious bug in the ransomware that makes the attacker unable to recover any encrypted files. Finally, we announce that decryption is still possible without paying the ransom, or to be more specific, only possible without paying the ransom.

Overview

There are quite a few differences between this Lorenz variant and what we have seen before. For instance, the ransom note used to be a HTML document called 'HELP_SECURITY_EVENT.html'. The ransom note of the new variant is a plain text document called 'HELP.TXT'.

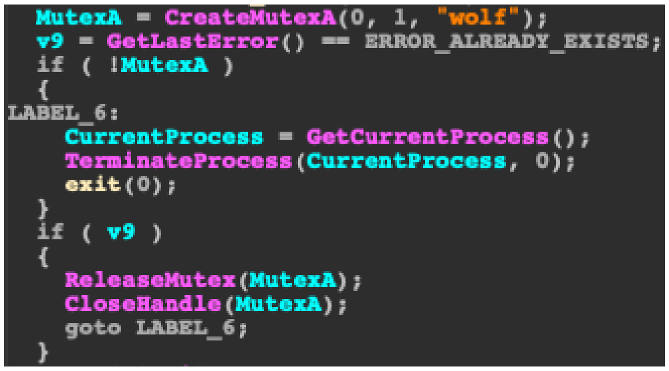
Like the previous one, this Lorenz variant also contacts a command & control server. The contact is initialized via HTTP on port 80 by sending a small POST request containing the infected system's computer name, internal IP-address and Windows OS version (e.g., 6.1 for Windows 7). This command & control communication might be used to provide basic usage statistics for the attackers.

Another difference is string encryption. In the first variant of Lorenz we analyzed, strings were embedded in the binary in plain text. This time, they are encrypted using a simple algorithm: they are XOR-encrypted using the single-byte key 0x6B and then Base64-encoded.

Unfortunately, this Lorenz variant contains a couple of serious bugs and mistakes. Unlike its predecessor, this variant does not create and check a mutex on startup. Furthermore, the encryption key needed to decrypt an affected file is not written, and therefore, decryption is virtually impossible.

Possible double encryption

It is common for ransomware to create a mutex on startup and terminate if the mutex already exists. This is a very important step, as it prohibits multiple instances from the ransomware to run simultaneously. The first variant of Lorenz we analyzed correctly created and checked a mutex called 'wolf'. The process would automatically terminate if that mutex already exists, as shown in the screenshot below.



```
  MutexA = CreateMutexA(0, 1, "wolf");
  v9 = GetLastError() == ERROR_ALREADY_EXISTS;
  if ( !MutexA )
  {
LABEL_6:
    CurrentProcess = GetCurrentProcess();
    TerminateProcess(CurrentProcess, 0);
    exit(0);
  }
  if ( v9 )
  {
    ReleaseMutex(MutexA);
    CloseHandle(MutexA);
    goto LABEL_6;
  }
```

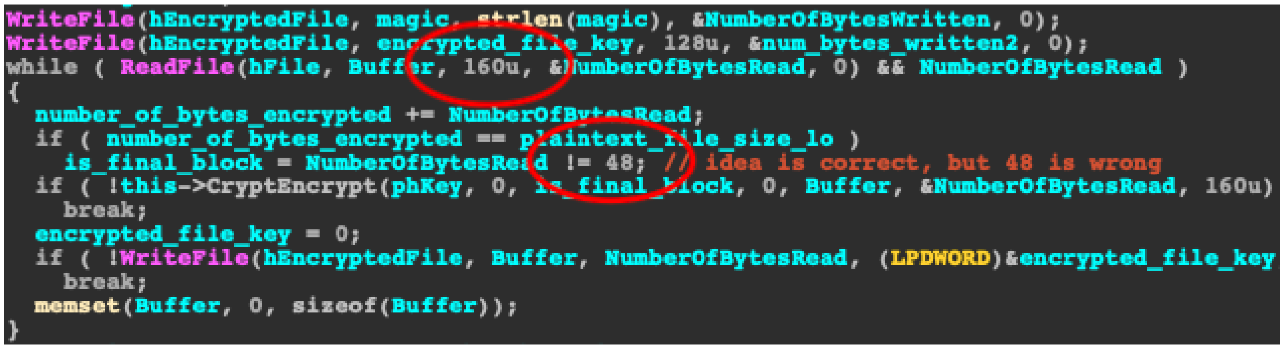
The recent variant of Lorenz we analyzed does not create or check a mutex. Therefore, multiple instances can be run simultaneously, allowing different instances to corrupt each other's encrypted files to a point where decryption would later be very difficult or even impossible.

File encryption

The core of the file encryption scheme has remained the same. Files are encrypted using a combination of RSA and AES encryption in CBC mode. A password is generated at random and used to derive the actual encryption key using the CryptDeriveKey function.

However, the way files are encrypted has changed. The previous variant encrypted every file whole, in blocks of 48 bytes. The encrypted file marker '.sz40' and RSA-encrypted password are then placed at the head of the file. The recent variant divides files into two categories: files smaller than 64368875 bytes (~ 61MiB), and files above this size. Small files are encrypted whole, but in blocks of 160 bytes. Large files are handled differently. Only the first 4000000000 bytes (~3.7GB) of these files are encrypted. Small files are encrypted side-by-side, because the encrypted file marker and encryption key are placed in the head of the file. That means, the encrypted file is created, and the original file is deleted after encryption. Large files are encrypted in-place, which means that every block is read, encrypted, and written back to the file. After encryption, the file is renamed to its encrypted counterpart.

As discussed above, files used to be encrypted in blocks of 48 bytes in the previous variant. The algorithm for encrypting small files in the new variant does this in blocks of 160 bytes. The authors might have altered the original encryption algorithm to now encrypt small files only and created a new algorithm to encrypt the large files. As shown in the screenshot below, the authors forgot to alter one instance of 48 to 160. In practice, 'is_final_block' will always be TRUE once the end of the file has been reached, unless the remainder is 48 bytes. If the remaining number of bytes (the final block smaller than or equal to 160) is not a multiple of 8, the last bytes of the encrypted file may contain invalid data.



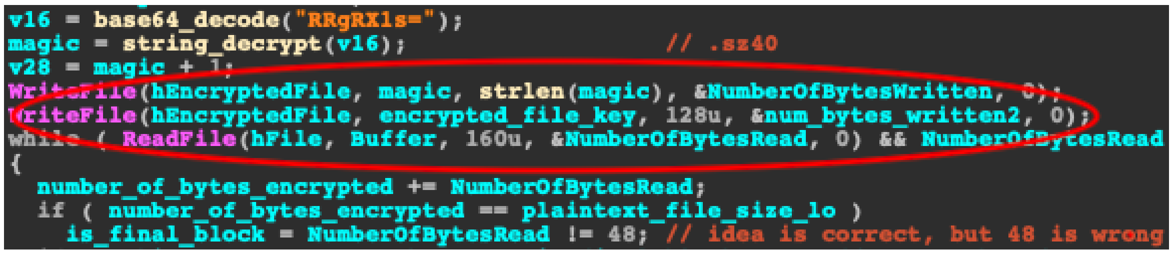
```
WriteFile(hEncryptedFile, magic, strlen(magic), &NumberOfBytesWritten, 0);
WriteFile(hEncryptedFile, encrypted_file_key, 128u, &num_bytes_written2, 0);
while ( ReadFile(hFile, Buffer, 160u, &NumberOfBytesRead, 0) && NumberOfBytesRead )
{
  number_of_bytes_encrypted += NumberOfBytesRead;
  if ( number_of_bytes_encrypted == plaintext_file_size_lo )
    is_final_block = NumberOfBytesRead != 48; // idea is correct, but 48 is wrong
  if ( !this->CryptEncrypt(phKey, 0, is_final_block, 0, Buffer, &NumberOfBytesRead, 160u)
    break;
  encrypted_file_key = 0;
  if ( !WriteFile(hEncryptedFile, Buffer, NumberOfBytesRead, (LPDWORD)&encrypted_file_key
    break;
  memset(Buffer, 0, sizeof(Buffer));
}
```

A serious mistake

For any ransomware to be successful, the ability to correctly recover the files that it encrypted is essential. If the attackers are unable to do so, they lose much of their leverage against the victim. The Lorenz authors made a serious mistake in the recent variant, rendering encrypted files un-decryptable.

To recover an encrypted file, we need the key that was used to encrypt it. Recall that Lorenz randomly generates a password to derive the actual AES encryption key. This encryption key is then encrypted using RSA with the public key embedded in the ransomware by the attackers, which is common practice. Additionally, this RSA-encrypted key is written to the head or tail of the encrypted file. After all, we need it in order to recover the original file contents later.

This is where the most serious mistake was made. The password is 40 characters long, but any RSA-1024 encrypted data requires 128 bytes (1024 / 8 bits) of memory space. The ransomware did not create a buffer large enough to fit these 128 bytes. As shown in the screenshot below, the encrypted file marker and RSA-encrypted key are supposed to be written to the encrypted file using the WriteFile function. Since WriteFile would read outside of the 40-byte buffer 'encrypted_file_key', it fails and returns the 'ERROR_INVALID_USER_BUFFER' error. The returned error is not checked, and the file continues to be encrypted.



```
v16 = base64_decode("RRgRX1s=");
magic = string_decrypt(v16);                    // .sz40
v28 = magic + 1;
WriteFile(hEncryptedFile, magic, strlen(magic), &NumberOfBytesWritten, 0);
WriteFile(hEncryptedFile, encrypted_file_key, 128u, &num_bytes_written2, 0);
while ( ReadFile(hFile, Buffer, 160u, &NumberOfBytesRead, 0) && NumberOfBytesRead
{
  number_of_bytes_encrypted += NumberOfBytesRead;
  if ( number_of_bytes_encrypted == plaintext_file_size_lo )
    is_final_block = NumberOfBytesRead != 48; // idea is correct, but 48 is wrong
```

The above screenshot was taken from the small-file encryption algorithm, but the large file algorithm suffers from the exact same issue. The result of this mistake is that encrypted files only contain the marker and encrypted contents. Without the key, recovering the file is virtually impossible.

Conclusion

For the previous Lorenz variant we analyzed, we concluded that decryption was possible without paying the ransom. Furthermore, we shed light on a bug that would destroy the last 48 bytes of some files. However, this new variant of Lorenz contains more serious issues, such as the destruction of files. Even though the 48-byte bug is fixed, we can safely conclude that this new variant is destructive.

Thankfully, decryption is still possible without paying the ransom. Have you have been hit by Lorenz? Don't pay the ransom, as in some cases it will not get your files back. Contact us to discuss how we may be able to help you recover your data without paying the ransom!

Indicators of Compromise

Indicator	Description
427713275e7c4515e3c577684a7c5f96e45771fee54f1c3162a2d6a2cc4cd76e	SHA-256 of new Lorenz variant sample
172[.]86[.]75[.]81	Command & control server