Decrypted: TaRRaK Ransomware

by Threat Research Team
June 6

The `TaRRaK` ransomware appeared in June of 2021. This ransomware contains many coding errors, so we decided to publish a small blog about them. Samples of this ransomware were spotted in our user base, so we also created a decryptor for this ransomware.

Behavior of the ransomware

The ransomware is written in .NET. The binary is very clean and contains no protections or obfuscations. When executed, the sample creates a mutex named `TaRRaK` in order to ensure that only one instance of the malware is executed. Also, an auto-start registry entry is created in order to execute the ransomware on every user login:

```
Windows Registry Editor Version 5.00

[HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run]
"TaRRaK"="\"<PATH-TO-SAMPLE>\""
```

The ransomware contains a list of 178 file types (extensions) that, when found, are encrypted:

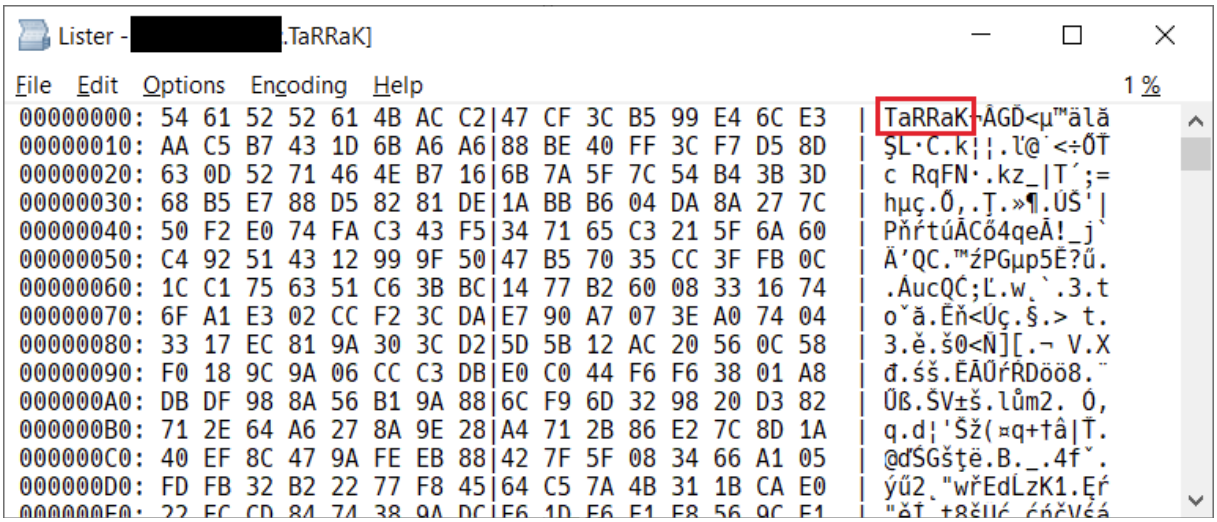
```
3ds 7z 7zip acc accdb ai aif apk asc asm asf asp aspx avi backup bak bat
bin bmp c cdr cer cfg cmd cpp crt crw cs csproj css csv cue db db3 dbf dcr
dds der dmg dng doc docm docx dotx dwg dxf dxg eps epub erf flac flv gif
gpg h html ico img iso java jpe jpeg jpg js json kdc key kml kmz litesql
log lua m3u m4a m4u m4v max mdb mdf mef mid mkv mov mp3 mp4 mpa mpeg mpg
mrw nef nrw obj odb odc odm odp ods odt orf p12 p7b p7c part pdb pdd pdf
pef pem pfx php plist png ppt pptm pptx ps ps1 psd pst ptx pub pri py pyc
r3d raf rar raw rb rm rtf rwl sav sh sln suo sql sqlite sqlite3 sqlitedb
sr2 srf srt srw svg swf tga thm tif tiff tmp torrent txt vbs vcf vlf vmx
vmdk vdi vob wav wma wmi wmv wpd wps x3f xlk xlm xls xlsb xlsm xlsx xml zip
```

The ransomware avoids folders containing one the following strings:

- `All Users\Microsoft\`

- `$Recycle.Bin`
- `:\Windows`
- `\Program Files`
- `Temporary Internet Files`
- `\Local\Microsoft\`
- `:\ProgramData\`

Encrypted files are given a new extension `.TaRRaK`. They also contain the TaRRaK signature at the beginning of the encrypted file:



File Encryption

Implementation of the encryption is a nice example of a buggy code:

```
// Token: 0x06000010 RID: 16 RVA: 0x00002694 File Offset: 0x00000894
private void EncryptFile(string path)
{
    //...

    byte[] array = new byte[0];
    int i = 0;
    while (i < 1)
    {
        try
        {
            // Attempt to read the entire files.
            // Fails when file size > 2GB
            array = File.ReadAllBytes(path);

            // Encrypt the content of the file.
            array = this._encrypter.Encrypt(array);

            i++;
            this._encrypted.Add(path);
        }
        catch
        {
            try
            {
                //
                // Reset file permissions
                //
            }
            catch
            {}
        }
    }
    try
    {
        // Write the encrypted data to file and rename it
        File.WriteAllBytes(path, array);
        File.Move(path, path + this._extension);
    }
    catch
    {}
```

First, the ransomware attempts to read the entire file to memory using `File.ReadAllBytes()`. This function has an internal limit – a maximum of `2 GB` of data can be loaded. In case the file is larger, the function throws an exception, which is then handled by the try-catch block. Unfortunately, the try-catch block only handles a permission-denied condition. So it adds an ACL entry granting full access to everyone and retries the read data operation. In case of any other error (read failure, sharing violation, out of memory, read from an offline file), the exception is raised again and the ransomware is stuck in an infinite loop.

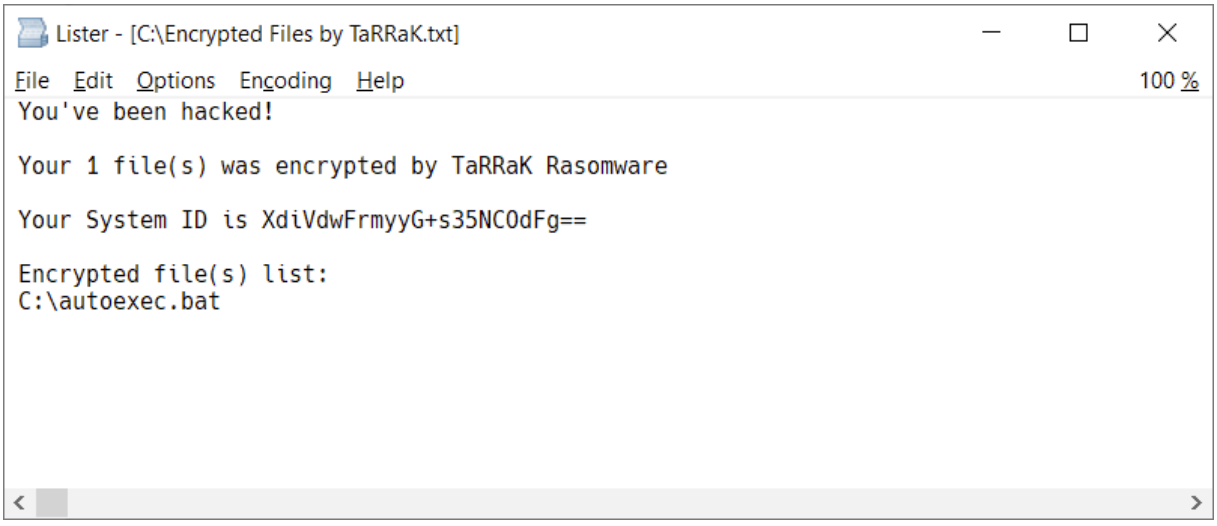
Even if the data load operation succeeds and the file data can be fit in memory, there's another catch. The `Encrypt` function converts the array of bytes to an array of 32-bit integers:

```
// Token: 0x06000002 RID: 2 RVA: 0x0000207C File Offset: 0x0000027C
public byte[] Encrypt(byte[] data)
{
    byte[] result;

    if (data.Length)
    {
        result = this.ToByteArray(this.Encrypt(this.ToUInt32Array(data, true),
                                               this.ToUInt32Array(this._key, false)
                                               ),
                                  false);
    }
    else
    {
        result = data;
    }
    return result;
}
```

So it allocates another block of memory with the same size as the file size. It then performs an encryption operation, using a custom encryption algorithm. Encrypted Uint32 array is converted to another array of bytes and written to the file. So in addition to the memory allocation for the original file data, two extra blocks are allocated. If any of the memory allocations fails, it throws an exception and the ransomware is again stuck in an infinite loop.

In the rare case when the encryption process finishes (no sharing violation or another error), the ransom note file named `Encrypted Files by TaRRaK.txt` is dropped to the root folder of each drive:



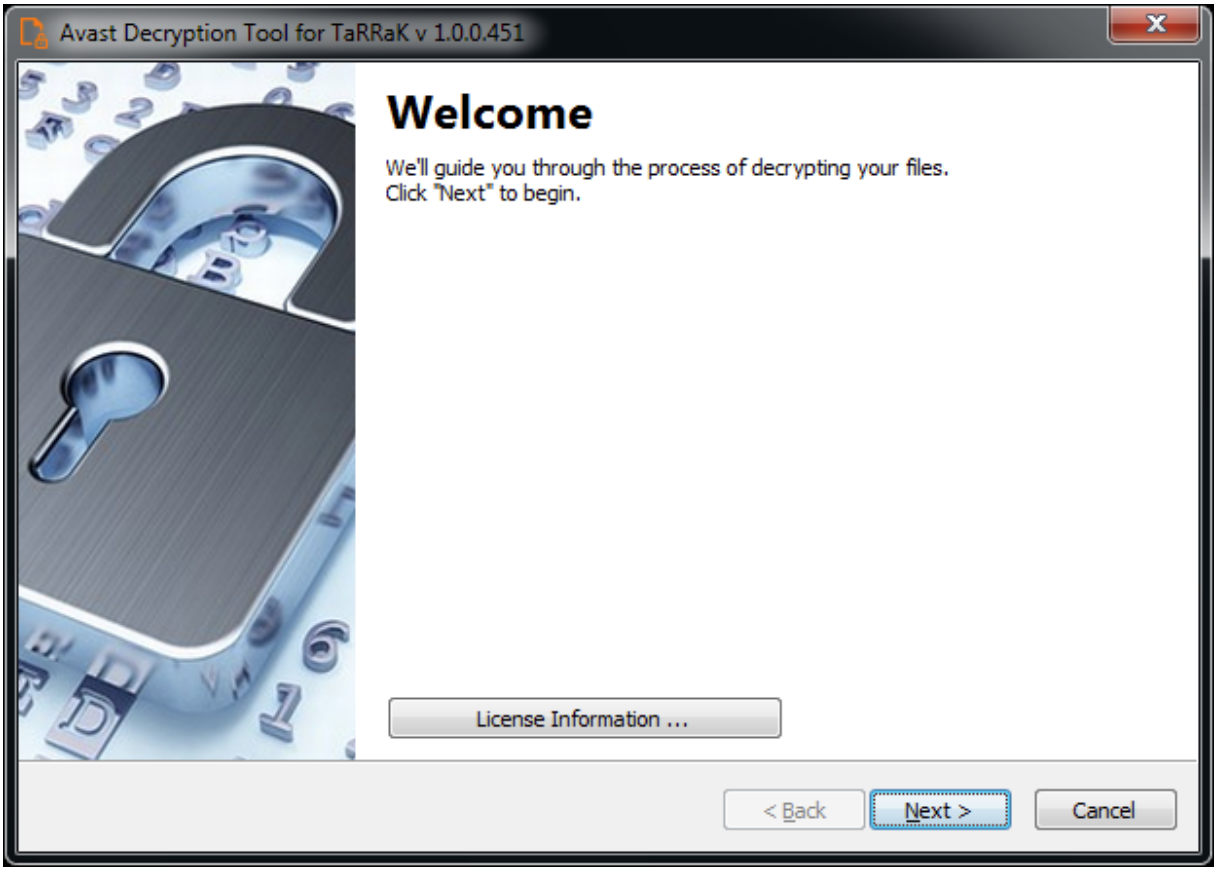
Files with the `.TaRRaK` extension are associated with their own icon:

Finally, desktop wallpaper is set to the following bitmap:

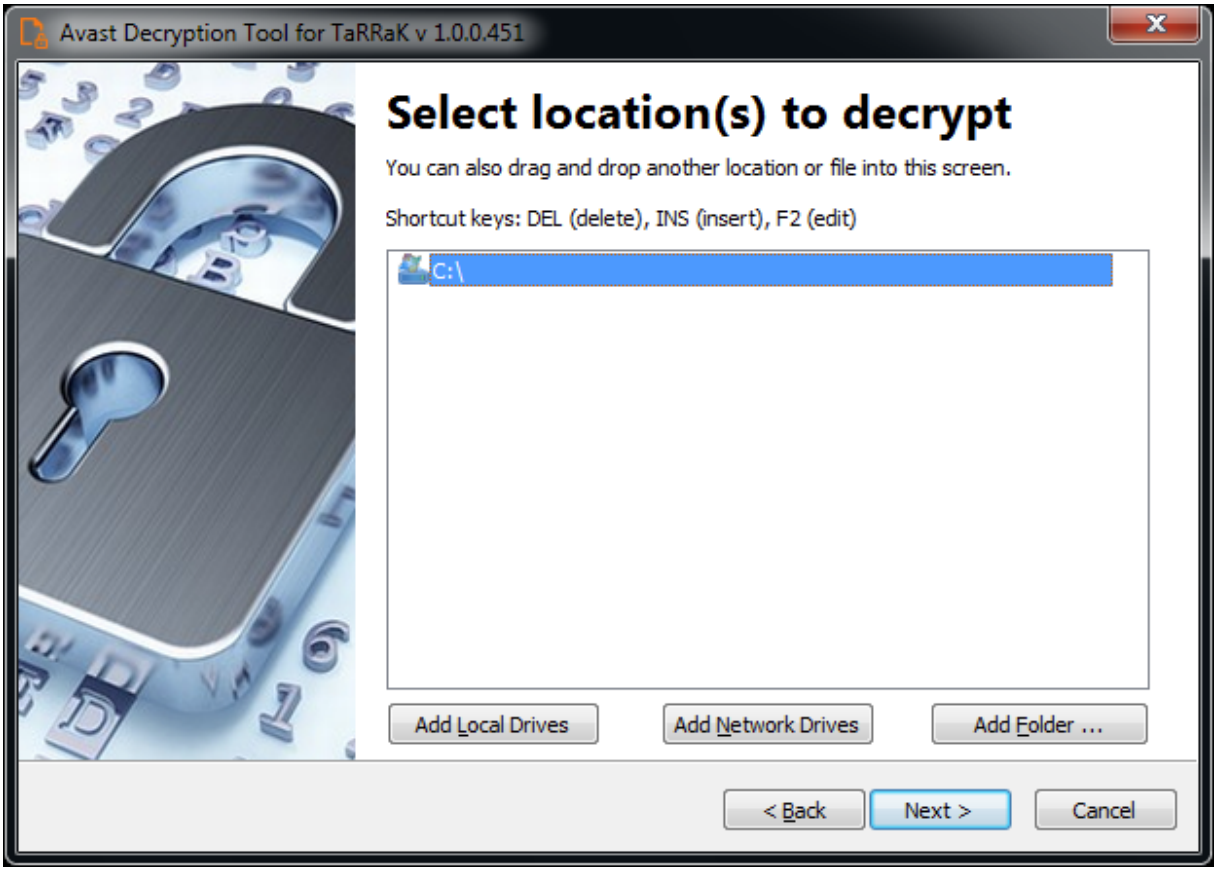
How to use the Avast decryptor to decrypt files encrypted by TaRRaK Ransomware

To decrypt your files, follow these steps:

1. You must be logged to the same user account like the one under which the files were encrypted.
2. Download the free Avast decryptor for 32-bit or 64-bit Windows.
3. Run the executable file. It starts in the form of a wizard, which leads you through the configuration of the decryption process.
4. On the initial page, you can read the license information, if you want, but you really only need to click “Next”



1. On the next page, select the list of locations you want to be searched and decrypted. By default, it contains a list of all local drives:



1. On the final page, you can opt-in to backup encrypted files. These backups may help if anything goes wrong during the decryption process. This option is turned on by default, which we recommend. After clicking “Decrypt”, the decryption process begins. Let the decryptor work and wait until it finishes decrypting all of your files.

IOCs

SHA256

```
00965b787655b23fa32ef2154d64ee9e4e505a42d70f5bb92d08d41467fb813d
47554d3ac4f61e223123845663c886b42016b4107e285b7da6a823c2f5050b86
aafa0f4d3106755e7e261d337d792d3c34fc820872fd6d1aade77b904762d212
af760d272c64a9258fab7f0f80aa2bba2a685772c79b1dec2ebf6f3b6738c823
```