#ShortAndMalicious — DarkGate

medium.com/@DCSO_CyTec/shortandmalicious-darkgate-d9102a457232

DCSO CyTec Blog September 19, 2023

DCSO CyTec Blog

--

A dark gate /
If you follow malware news, you've surely heard of the DarkGate loader by now. Using a rather exotic combination of AutoIt automation scripts and the Delphi programming language, its purpose is malware delivery and credential theft.

Given its apparent popularity with miscreants currently, DCSO's CSIRT decided to investigate.

While most of its capabilities have been covered in great detail already such as here, here and here, we noticed during our analysis that the key log encryption apparently had evolved since the great write-up by Strotz Friedberg published in early August 2023.

Thus, for this #ShortAndMalicious we'll show the new, convoluted process of deriving the key log encryption key (and bot ID!), and we also provide updated tooling for the community over on our GitHub.

Blog post authored by

Key log files

One of the features of DarkGate is key logging. Key strokes are recorded into encrypted log files and regularly uploaded to the C2 server. The files are encrypted using AES, but while earlier DarkGate versions appear to have used a fixed key, current versions have moved away from it.

Instead, the key is derived from the bot ID now, which in turn is derived from system specs, with intermediate steps involving some hashing and custom encoding.

For reasons more clear after reading the following, we're releasing 3 separate tools to handle encrypted key log files:

- A Windows tool to read the necessary system specs needed to calculate the bot ID
- A Python script to use that info to derive the bot ID and AES key
- A Delphi tool to decrypt log files with a key from the previous step and an encrypted log file

Calculating the AES key

In order to calculate the AES key used for key log files, we first need to determine the bot ID (aka hwid) of the system for which we want to decrypt the key log files.

The outline of the bot ID generation is as follows:

- Fetch system information
- Calculate MD5 sum of system information
- Use a custom encoding for the MD5 digest

Bot ID generation

For the bot ID generation, DarkGate gathers five pieces of information:

1. Computer name
2. User name
3. Processor information from the registry value `HKLM\HARDWARE\DESCRIPTION\System\CentralProcessor\0\ProcessorNameString`

4. Windows product ID from the registry value `HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProductId`
5. Number of processors via API `GetSystemInfo`

The number of processors is appended to the processor info in the form of "@ x Cores", and then it combines all these into a string and calculates the MD5 digest for it:

```
digest = MD5(product_id+processor+user+computer)
```

And finally, it takes the 16 byte MD5 digest and encodes it using a custom alphabet by using it as a lookup table nibble-wise (half bytes, that is). Reimplemented in Python, it looks like this:

```
def custom_encode(s):
    lookup = b"abcdefKhABCDEFGH"



    out = bytearray()
    for i in range(len(s)):
        upper_nibble = (s[i] & 0xF0) >> 4
        lower_nibble = s[i] & 0x0F

        out.append(lookup[upper_nibble])        out.append(lookup[lower_nibble])
return out
```

AES key derivation

For the AES key derivation, then, we need the following ingredients:

- Hardcoded string "mainhw"
- Bot ID as calculated before
- Config value `internal_mutex`

The latter can be extracted using the most excellent DarkGate config extractor published by Fabian Marquardt over on the Telekom Security blog.

With these pieces of information, the calculation is as follows:

Calculate and encode the MD5 digest as with the bot ID:

```
digest = MD5("mainhw"+bot_id+internal_mutex)encoded = custom_encode(digest)
```

Then finally, the AES key used by DarkGate to encrypt the key log files is the first 7 letters, lowercased:

```
aes_key = encoded[:7].lower()
```

So in summary the complete process looks like this:

```
digest = MD5(product_id+processor+user+computer)bot_id = custom_encode(digest)digest2
= MD5("mainhw"+bot_id+internal_mutex)encoded = custom_encode(digest2)aes_key =
encoded[:7].lower()
```

Decrypting log files

Now that we have the AES key, we want to decrypt log files, but it turns out it is not that simple. Long story short, the AES implementation used is incompatible to most readily available implementations used in, say, Python.

Or in more detail: DarkGate uses the function `EncryptString` offered by the Delphi library dcpcrypt. `EncryptString` has three quirks though:

1. It feeds the given key to SHA1 then uses the 160 bit output as key value. AES is only standardized for 128/192/256 bit key sizes, but Rijndael, the algorithm behind the AES standard can actually use 160 bit sizes, so , this is using Rijndael and not AES
2. It uses a custom cipher mode to turn the Rijndael block cipher into a stream cipher which it calls mode.
3. If that wasn't enough, that dcpcrypt produces key schedules incompatible with other Rijndael implementations for non-standard key sizes, such as — unfortunately — the 160 bit used in DarkGate.

Given all these hurdles, we followed the example of AON's Cyber Labs and simply developed a decryption tool in Delphi using the same library and it's… quirks.

Our tool takes as input a 7 letter AES key as generated by the aforementioned algorithm, as well as an input file path and an output file path, and then simply decrypts the file.

Below is an example of a decrypted key log file. The hex-encoded string between the `encwindow` markers is the UTF16-encoded window title for which the input was captured, followed by the input captured for this window:

Decrypted key log file
Head over to our GitHub for the tools!