Open the DARKGATE – Brute Forcing DARKGATE Encodings

By Sean Straw

Published: 2024-01-18 · Archived: 2026-04-05 17:29:29 UTC

Key Takeaways

- Kroll has observed a recent shift in the base64 encoding used for DARKGATE.
- The base64 alphabet is now randomized based on characteristics of the victim system.
- A weakness in the seed value randomness makes the new alphabet trivial to brute force.
- The discovered alphabet can be used to decode the on-disk configuration and keylogging outputs.
- The keylogger output files contain the keystrokes stolen by DARKGATE. Examiners can analyze these files to identify potentially stolen passwords or other at-risk information.

Summary

DARKGATE is Windows-based malware that is sold on the dark web. DARKGATE is a fully functional backdoor that can steal browser information, drop additional payloads, and steal keystrokes. Kroll previously noted DARKGATE's distribution via Teams.

When the DARKGATE payload runs on a victim system, it creates a randomly named folder within C:\ProgramData that contains encoded files. Within the randomly named folder is a short configuration file and the output of keystrokes logged on the system. Both the keylogging output and the configuration files are encoded using a custom base64 alphabet. In previous versions, the encoding used a hardcoded, nonstandard alphabet.

Kroll recently analyzed newer versions of DARKGATE, specifically 5.2.3, which randomly shuffles the nonstandard alphabet in use. Kroll identified a weakness in this shuffling that trivializes guessing the correct alphabet for a system without needing the hardware ID.

This analysis enables forensic analysts to decode the configuration and keylogger files without needing to first determine the hardware ID. The keylogger output files contain keystrokes stolen by DARKGATE, which can include typed passwords, composed emails and other sensitive information.

DARKGATE's Created Files

Below is an example folder structure created by DARKGATE:

- .\ProgramData\hgehakb\Autoit3.exe (Legitimate AutoIT executable)
- .\ProgramData\hgehakb\abbhebe.au3 (Loader script)
- .\ProgramData\hgehakb\bebdbhk\08-12-2023.log (Encoded keylogger captures)
- .\ProgramData\hgehakb\bebdbhk\cffhbdd (Encoded stored configuration)

DARKGATE's Hardware ID Generation

When DARKGATE runs, it generates a unique hardware ID. This value is comprised of several concatenated system attributes:

- The Windows version
- The product ID
- The processor name and number of cores
- The username
- The system hostname

These values are combined and hashed using MD5. Rather than using the standard hexadecimal representation for an MD5 hash, DARKGATE uses a substituted alphabet consisting of abcdefKhABCDEFGH, as illustrated in Figure 1.

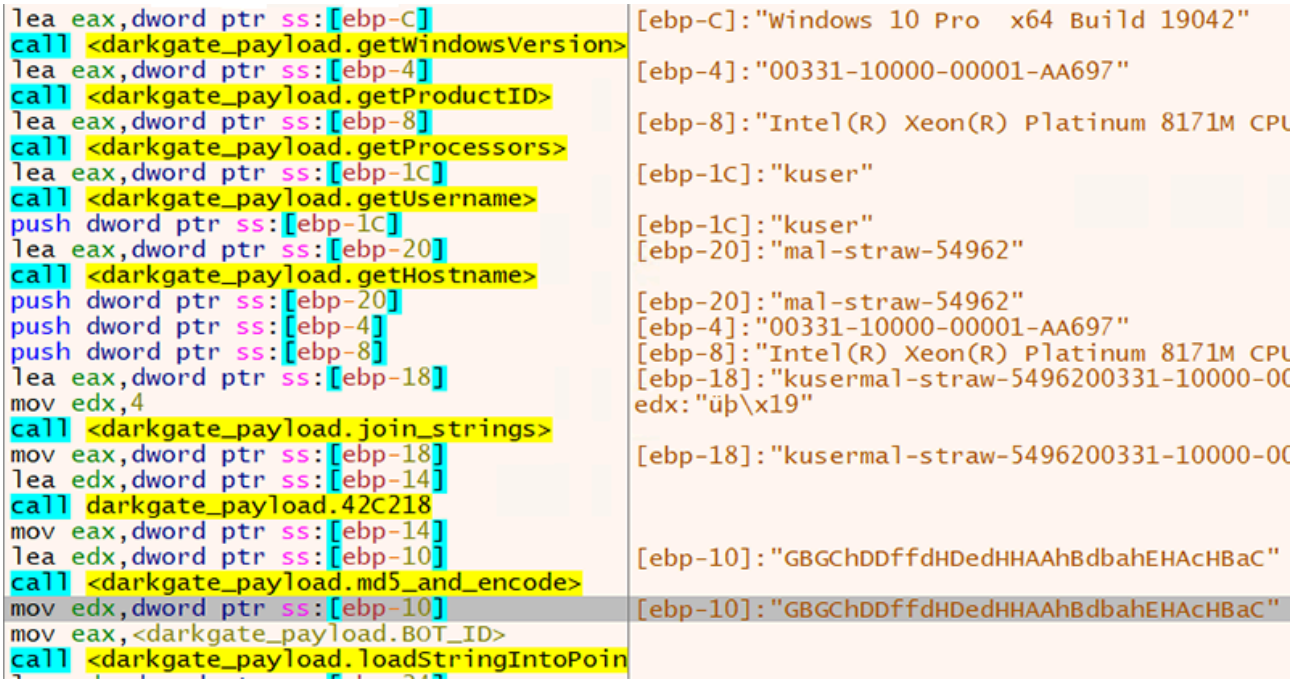

Figure 1. Generation of the bot ID

The DARKGATE Shuffle

The version of DARKGATE that was analyzed shuffles the base64 alphabet in use at the initialization of the program. DARKGATE swaps the last character with a random character before it, moving from back to front in the alphabet. The shuffling occurs as follows:

- The hardware ID is used to create a seed value for a pseudorandom number generator. The seed value is generated by adding each byte of the 32-byte ID together. This becomes the seed value.
- A counter starting at the last digit of the hardcoded base64 alphabet is set.
- The pseudorandom number generator multiplies the seed by 0x8088405 and adds 1, storing this as the new SRAND value.

- The new seed value is multiplied by the length of the counter and shifted 32 bits right. This effectively chooses a pseudorandom number between 0 and the counter (non-inclusive).
- The alphabet character at the randomly chosen number and the counter are swapped.
- The counter is decremented. Execution then returns to step 3 until the counter reaches 0.

As base64 has an alphabet length of 64, this leaves 64!, or roughly 1.2 × 10^89, possible alphabets.

The alphabet shuffling is straightforward, as shown in Figure 2.

```
                Get a random offset between the start and the iterator
                LAB_004lalef                                    XREF[1]:
8b c5               MOV         EAX,EBP
e8 0e 8f            CALL        pseudoRandRange
fe ff
8b f8               MOV         EDI,EAX
47                  INC         EDI
8b 06               MOV         EAX,dword ptr [ESI]
8a 5c 28 ff         MOV         BL,byte ptr [EAX + EBP*0x1 + -0x1]
8b c6               MOV         EAX,ESI
e8 ea a6            CALL        getStrLen
fe ff
                Swap random offset character with iterator index
8b 16               MOV         EDX,dword ptr [ESI]
8a 54 3a ff         MOV         DL,byte ptr [EDX + EDI*0x1 + -0x1]
88 54 28 ff         MOV         byte ptr [EAX + EBP*0x1 + -0x1],DL
8b c6               MOV         EAX,ESI
e8 d9 a6            CALL        getStrLen
fe ff
                Move temporary value back to iterator index and decrement
88 5c 38 ff         MOV         byte ptr [EAX + EDI*0x1 + -0x1],BL
4d                  DEC         EBP
83 fd 01            CMP         EBP,0x1
75 ce               JNZ         LAB_004lalef

                end_loop                                        XREF[1]:
```

Figure 2. The DARKGATE alphabet shuffle implementation

The “hash” of the bytes occurs right before the alphabet shuffling (see Figure 3).

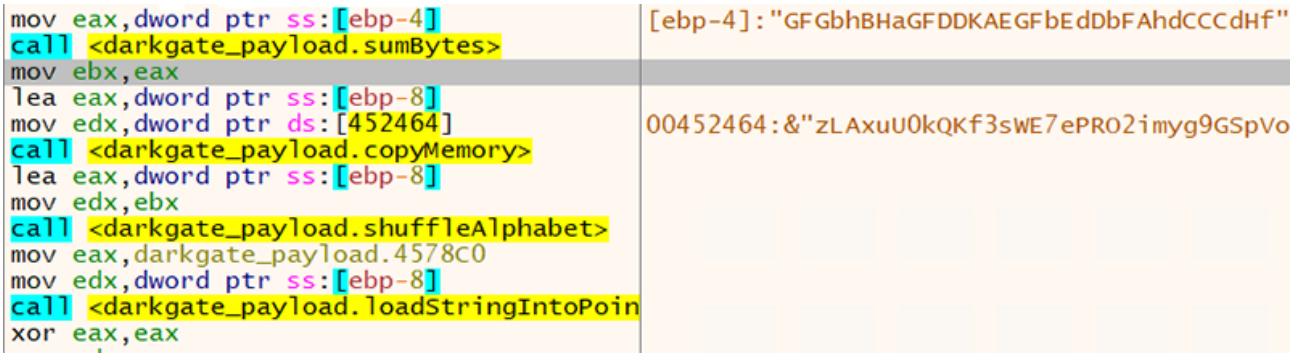

Figure 3. Summing the hardware ID bytes just before shuffling the alphabet

Weak Seed Generation

The flaw in the DARKGATE alphabet shuffle exists within the use of the sum of the hardware ID as a seed. As discussed previously, the DARKGATE hardware ID is a 32-byte ascii representation of an MD5 sum. MD5 is a cryptographic hash, and the distribution of each byte is random. These bytes are then summed.

Despite the randomness of the hardware ID, there are specific lowest and highest possible byte values. In the sample Kroll analyzed, the lowest byte value in the hardware ID is A, or 0x41. The highest possible value is h, or 0x68. These represent the lowest and highest value ascii characters in the MD5 alphabet.

Therefore, the lowest possible seed value is 0x20 * 0x41 (32 A's), or 0x820 (2080 in decimal), while the highest possible seed value is 0x20 * 0x68 (32 h's), or 0xD00 (3328 in decimal). This leaves us with a total of 0xD00 - 0x820 = 0x4E0 (1248 in decimal) possible custom base64 alphabets—very easy to brute force!

Putting It into Practice

The DARKGATE shuffle can be reimplemented in Python. Then, by iterating through all possible combinations, base64 decoding of each alphabet can be attempted to look for values that decode properly.

```
def shuffle(alphabet: str, seed: int) -> str:
   """Python version of the DARKGATE shuffle"""
   alphabet = list(alphabet)
   for i in range(len(alphabet), 0, -1):
       seed = (seed * 0x8088405 + 1) & 0xFFFFFFFF
       rand_val = (i * seed) >> 32
       alphabet[rand_val], alphabet[i-1] = alphabet[i-1], alphabet[rand_val]
   return ''.join(alphabet)
```

While Python's base64 library does not include an option to use a nonstandard alphabet, use of the translate function simplifies the decoding. This precludes the need to reimplement the base64 encoding algorithm.

```
b64decode(ciphertext.translate(str.maketrans(custom_alphabet, BASE64_ALPHABET)))
```

The output shows the decoded alphabet.

```
 $ py .\darkgate_decode.py .\encoded_config

MIN_SEED: 2080
MAX_SEED: 3328
-->Alphabet:
82k+YQg3xdZJbhK40oOH5nqsUp6XC1fmM7EvRFISPtiATlzLjruGaWwDyB9=NeVc
-->Decoded:
domains=GkPdpxZB35LtSI9HV0WXS8PtSIcjGmcX34WBSedf
epoch=1701885746
puerto=2351
version=5.2.3
hwid=GBGChDDffdHDedHHAAhBdbahEHAcHBaC
```

The alphabet can then be used to decode .log files using a tool such as CyberChef, see Figure 4. As some of the logged keystrokes can end up as nonstandard characters, using the provided Python script is not recommended.

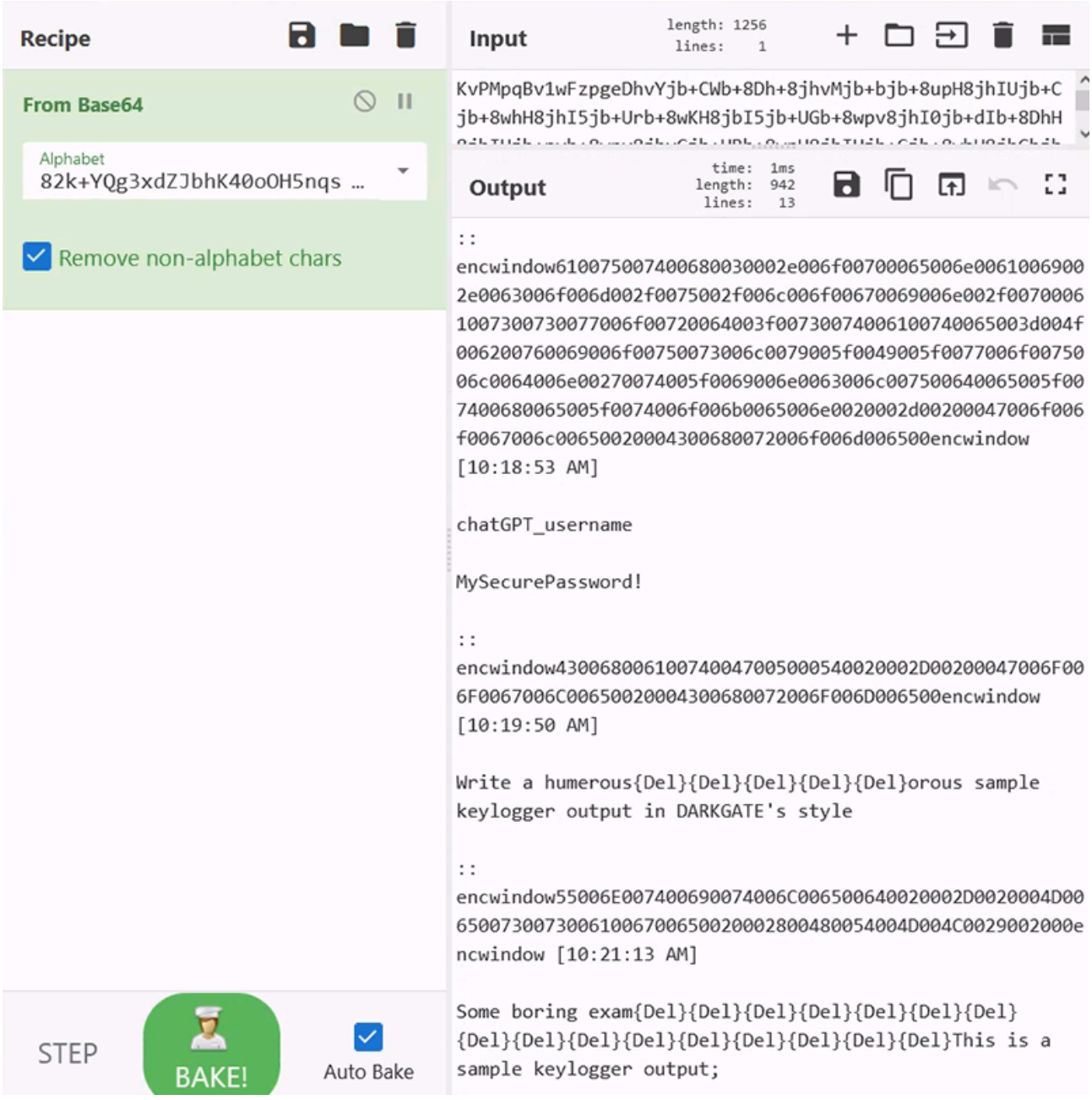

Figure 4. Decoding an example keylogger output in CyberChef

Within the keylogger output shown above, “encwindow” bookends the hex-encoded Window name of the program from which the keystroke was logged. In the above example, the windows are the OpenAI authentication link, “ChatGPT – Google Chrome” and “Untitled – Message (HTML).”

Script

Below is a script to brute force DARKGATE configuration files. Note that different samples may have different hardcoded base64 alphabets or MD5 alphabets. If so, the corresponding values should be modified.

```
 """darkgate_decode.py

Author: sean.straw@kroll.com

Decode a DARKGATE configuration file, identifying the alphabet in use.
Recommended this be run against a configuration file.
Use CyberChef or a similar tool to decode .log files with the identified alphabet.

Depending on the DARKGATE sample, you may need to modify the HWID alphabet.
You may also need to modify the hardcoded base64 alphabet.

Usage:
    py darkgate_decode.py <file>

Example:
    $ py darkgate_decode.py .\bekaeae

    py .\decode.py .\encoded_config
    MIN_SEED: 2080
    MAX_SEED: 3328
    -->Alphabet:
    82k+YQg3xdZJbhK40oOH5nqsUp6XC1fmM7EvRFISPtiATlzLjruGaWwDyB9=NeVc
    -->Decoded:
    domains=GkPdpxZB35LtSI9HV0WXS8PtSIcjGmcX34WBSedf
    epoch=1701885746
    puerto=2351
    version=5.2.3
    hwid=GBGChDDffdHDedHHAAhBdbahEHAcHBaC

The "domains" value is encoded with the hardcoded base64 alphabet, NOT the shuffled one.
"""

from typing import List, Tuple
import base64
import sys
from pathlib import Path

ORIGINAL_ALPHABET = r'zLAxuU0kQKf3sWE7ePRO2imyg9GSpVoYC6rhlX48ZHnvjJDBNFtMd1I5acwbqT+='
BASE64_ALPHABET = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/'

def shuffle(alphabet: str, seed: int) -> str:
    """Python version of the DARKGATE shuffle"""
    alphabet = list(alphabet)
    for i in range(len(alphabet), 0, -1):
        seed = (seed * 0x8088405 + 1) & 0xFFFFFFFF
        rand_val = (i * seed) >> 32
        alphabet[rand_val], alphabet[i-1] = alphabet[i-1], alphabet[rand_val]
    return ''.join(alphabet)

HWID_ALPHABET = [ord(x) for x in "abcdefKhABCDEFGH"]
MIN_SEED = min(HWID_ALPHABET) * 32
MAX_SEED = max(HWID_ALPHABET) * 32
print(f"MIN_SEED: {MIN_SEED}")
print(f"MAX_SEED: {MAX_SEED}")
ALL_ALPHABETS = [shuffle(ORIGINAL_ALPHABET, i) for i in range(MIN_SEED, MAX_SEED+1)]
ALL_ALPHABETS.append(ORIGINAL_ALPHABET)
```

```
def force(payload: str) -> List[Tuple[str, str]]:
    """Iterate through and find all possible alphabet solutions"""
    res = []
    for alphabet in ALL_ALPHABETS:
        t = payload.translate(str.maketrans(alphabet, BASE64_ALPHABET))
        # Add extra padding. The base64 module will trim it if not needed
        t += "==="
        try:
            t = base64.b64decode(t)
        except base64.binascii.Error:
            continue
        try:
            t = t.decode('utf-8')
        except ValueError:
            continue
        else:
            res.append((alphabet, t))
    return res

encoded_text = Path(sys.argv[1]).read_text(encoding='ascii')

possible_alphabets = force(encoded_text)
if len(possible_alphabets) < 1:
    print("Err: no alphabets found")
else:
    for (alphabet_, decoded) in possible_alphabets:
        print("-->Alphabet:")
        print(alphabet_)
        print("-->Decoded:")
        print(decoded)
```

Source: https://www.kroll.com/en/insights/publications/cyber/brute-forcing-darkgate-encodings