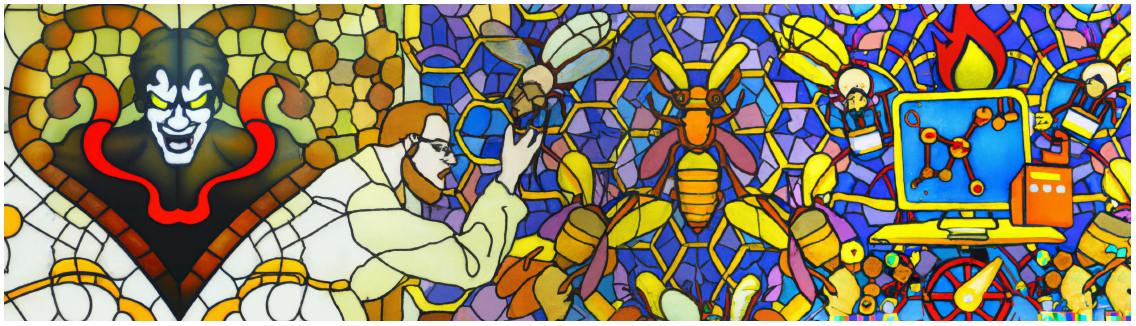
BumbleBee notes 🐝

By pbo

Published: 2023-03-28 · Archived: 2026-04-05 13:32:33 UTC

BumbleBee is categorized as a **Loader**, the malware is used by Initial Access Brokers to gain access in targeted companies. This article aims to summarizing the different TTPs observed in campaigns distributing BumbleBee and provides a script to extract its configuration.



TL;DR BumbleBee

The loader delivers diverse payloads (*e.g: Cobalt Strike, ransomware, etc*), the operators of BumbleBee have been named *EXOTIC LILY* by the TAG in a report published in March 2022. Google TAG article mentionned BumbleBee Loader (*e.g:* The user-agent set to bumblebee, hence dubbed BUMBLEBEE. https://blog.google/threat-analysis-group/exposing-initial-access-broker-ties-conti/ Moreover, similarities with other loaders in terms of operation have been noticed notably with IcedID and Emotet. Code similarty (*hook installation*) with Trickbot have been observed and explained in the post The chronicles of Bumblebee: The Hook, the Bee, and the Trickbot connection. The malware is well documented by now (*March 2023*) as evidenced by the number of reports on malpedia.

BumbleBee capabilities

The malware has a custom unpacking mechanism, it manipulates hooks to setup its execution chain, the loader uses multiple environment detection techniques because of the complete integration of the project al-khaser al-khaser is a PoC “malware” application with good intentions that aims to stress your anti-malware system. It performs a bunch of common malware tricks with the goal of seeing if you stay under the radar. . It communicates with its command and control over HTTP. Since August 2022 the malware embeds a list of IP addresses in its configuration, some of them are legitimate IP addresses, this technique is also used by other malware such as Emotet and Trickbot.

BumbleBee command and control IP addresses, port and the bot (or botnet) identifier are stored in the `.data` section, obfuscated with the *RC4* encryption algorithm. A script to extract and deobfuscate them is provided at the end of this post.

Campaigns file format

First malspam campaign which delivered BumbleBee contains a web link to a protected ZIP archive.

1. The archive contains an ISO file;
2. The ISO contains a LNK file and a DLL file;
3. The LNK executes `rundll32.exe` to invoke the embedded DLL;

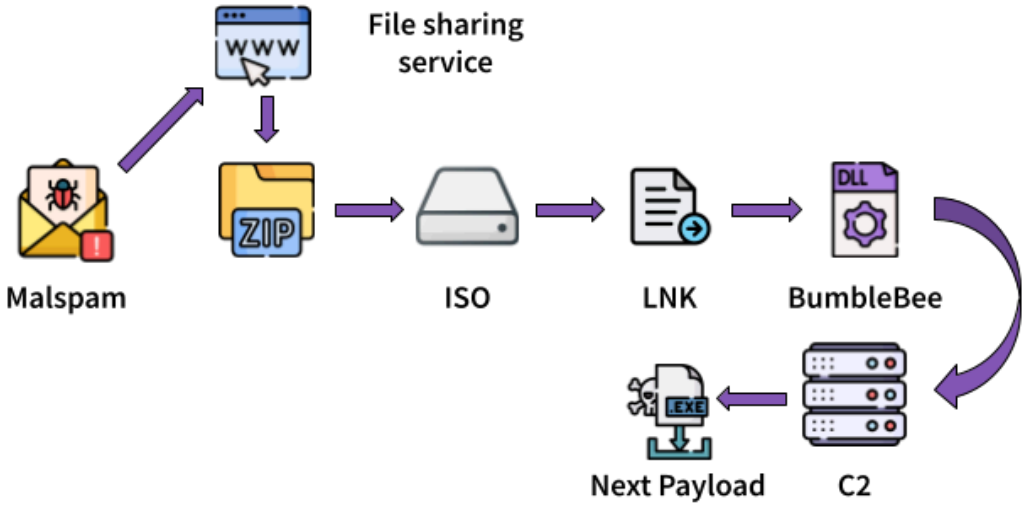


Figure 1: BumbleBee infection chain with ISO file

This model of campaign was used for months. During the summer of 2022, actors updated the disk image format from ISO to VHD. Content of disk image (*VHD*) changed too, the DLL is no more stored as a file, but it is embed obfuscated in a PowerShell script. The script is executed by the LNK with the **execution policy** set to **bypass**. The BumbleBee's DLL is stored in the PowerShell script in obfuscated strings (*e.g:* `$elem30=$elem30.$casda.Invoke(0,"H")`). After strings replacement, the base64 encoded variable is decoded, decrompressed (`ungzip`) and invoked (*e.g:* `scriptPath | iex`).

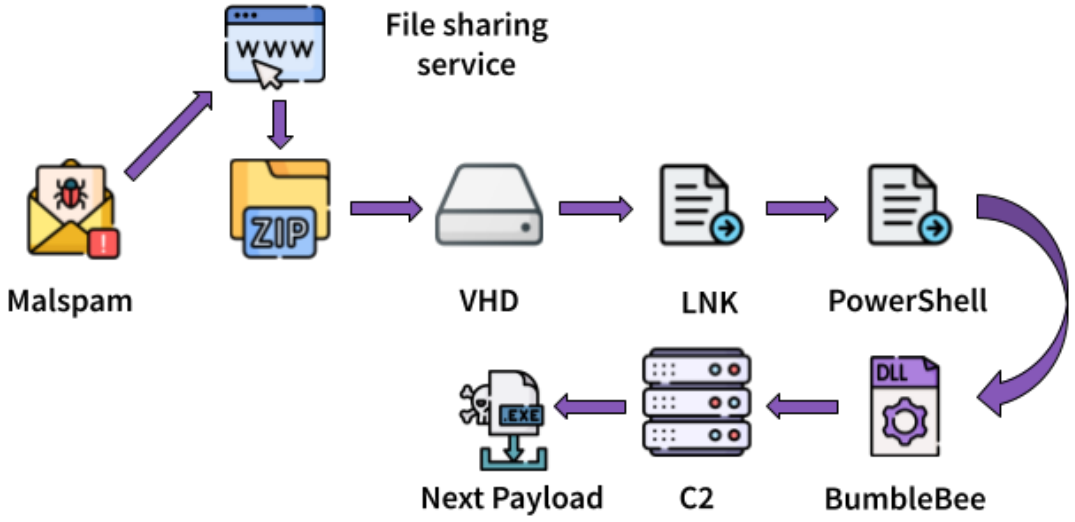


Figure 2: BumbleBee infection chain with VHD file

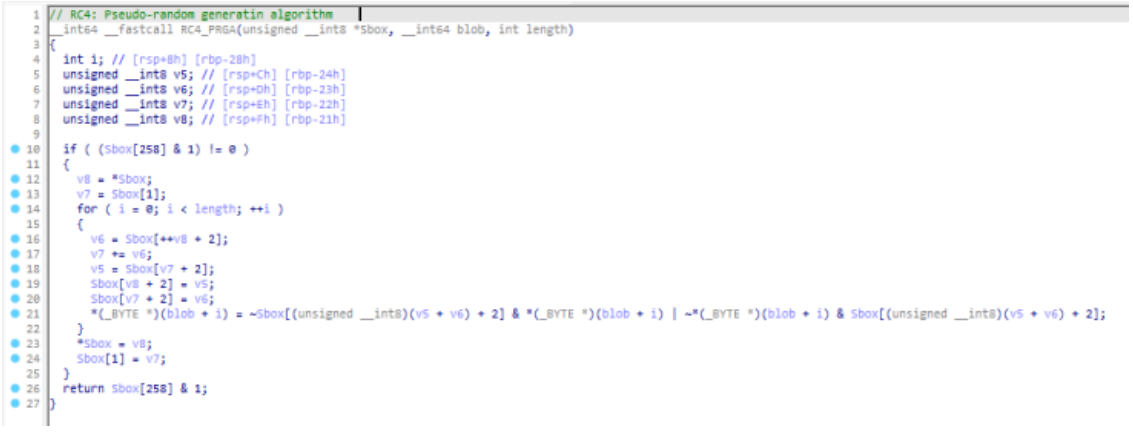
NB: File sharing service used to deliver BumbleBee change regulary e.g.: WeTransfer, Onedrive, Smash, etc. Details of a campaign using onedrive file sharing website are written in the article: Bumblebee DocuSign Campaign.

Examples IOCs:

- ISO: SHA-256: 8695f4936f2942d322e2936106f78144f91602c7acace080e48c97e97b888377
- VHD: SHA-256: e9a1ce3417838013412f81425ef74a37608754586722e00cacb333ba88eb9aa7

As introduced above, the configuration is stored encrypted with the RC4 algorithm. RC4: Rivest Cipher 4, also known as ARC4: https://en.wikipedia.org/wiki/RC4 The key is in cleartext in the binary and its length is repeatedly (for BumbleBee case) fixed to 10 characters.

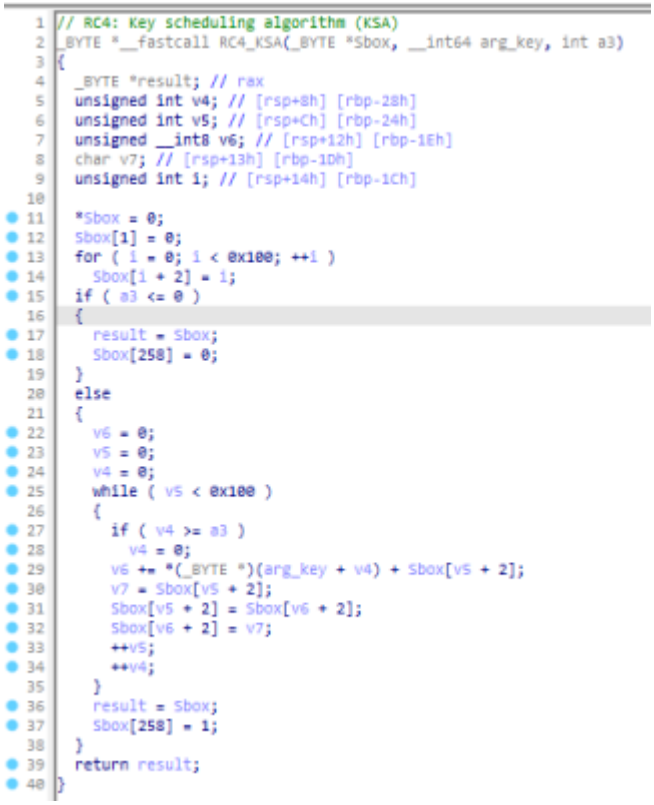
Here is the two functions that implement RC4 algorithm in BumbleBee:



```
// RC4: Pseudo-random generatin algorithm
__int64 __fastcall RC4_PRGA(unsigned __int8 *Sbox, __int64 blob, int length)
{
  int i; // [rsp+8h] [rbp-28h]
  unsigned __int8 v5; // [rsp+Ch] [rbp-24h]
  unsigned __int8 v6; // [rsp+Dh] [rbp-23h]
  unsigned __int8 v7; // [rsp+Eh] [rbp-22h]
  unsigned __int8 v8; // [rsp+Fh] [rbp-21h]

  if ( (Sbox[258] & 1) != 0 )
  {
    v8 = *Sbox;
    v7 = Sbox[1];
    for ( i = 0; i < length; ++i )
    {
      v6 = Sbox[++v8 + 2];
      v7 += v6;
      v5 = Sbox[v7 + 2];
      Sbox[v8 + 2] = v5;
      Sbox[v7 + 2] = v6;
      *(_BYTE *)(blob + i) = ~Sbox[(unsigned __int8)(v5 + v6) + 2] & *(_BYTE *)(blob + i) | ~*(_BYTE *)(blob + i) & Sbox[(unsigned __int8)(v5 + v6) + 2];
    }
    *Sbox = v8;
    Sbox[1] = v7;
  }
  return Sbox[258] & 1;
}
```

Figure 3: BumbleBee implemenation of PRGA of RC4 algorithm

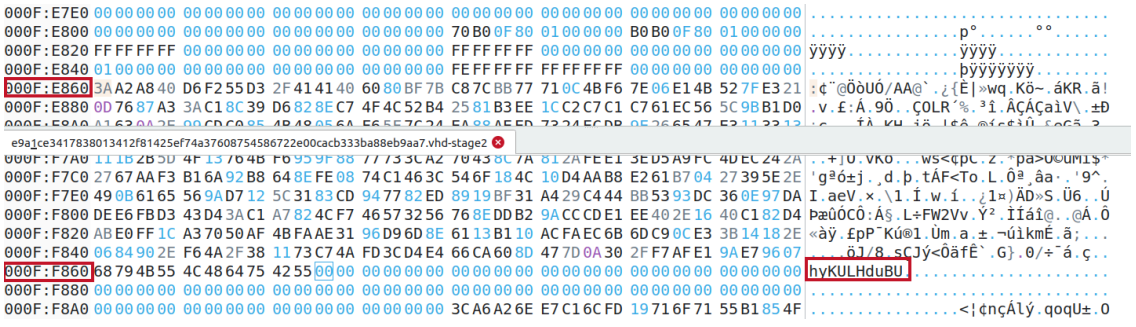


```
// RC4: Key scheduling algorithm (KSA)
_BYTE *__fastcall RC4_KSA(_BYTE *Sbox, __int64 arg_key, int a3)
{
  _BYTE *result; // rax
  unsigned int v4; // [rsp+8h] [rbp-28h]
  unsigned int v5; // [rsp+Ch] [rbp-24h]
  unsigned __int8 v6; // [rsp+12h] [rbp-1Eh]
  char v7; // [rsp+13h] [rbp-1Dh]
  unsigned int i; // [rsp+14h] [rbp-1Ch]

  *Sbox = 0;
  Sbox[1] = 0;
  for ( i = 0; i < 0x100; ++i )
    Sbox[i + 2] = i;
  if ( a3 <= 0 )
  {
    result = Sbox;
    Sbox[258] = 0;
  }
  else
  {
    v6 = 0;
    v5 = 0;
    v4 = 0;
    while ( v5 < 0x100 )
    {
      if ( v4 >= a3 )
        v4 = 0;
      v6 += *(_BYTE *)(arg_key + v4) + Sbox[v5 + 2];
      v7 = Sbox[v5 + 2];
      Sbox[v5 + 2] = Sbox[v6 + 2];
      Sbox[v6 + 2] = v7;
      ++v5;
      ++v4;
    }
    result = Sbox;
    Sbox[258] = 1;
  }
  return result;
}
```

Figure 4: BumbleBee implementation of KSA of RC4 algorithm

The key is stored at the end of the blob of data containing the encrypted list of IP addresses. After analysing few samples of BumbleBee, it appears that the blob of data containing the IP addresses is always **4105 bytes** long (*plus one null byte*) which is a pattern to look for in the DLL for a C2 extractor.



```
000F:E7E0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 ................................
000F:E800 00000000 00000000 00000000 00000000 70B00F80 01000000 B0B00F80 01000000 ............p°......°°......
000F:E820 FFFFFFFF 00000000 00000000 00000000 FFFFFFFF 00000000 00000000 00000000 ÿÿÿÿ............ÿÿÿÿ............
000F:E840 01000000 00000000 00000000 00000000 FEFFFFFF FFFFFFFF 00000000 00000000 ................þÿÿÿÿÿÿÿ........
000F:E860 3AA2A840 D6F255D3 2F414140 6080BF7B C87CBB77 710C4BF6 7E06E14B 527FE321 :¢¨@ÖòUÓ/AA@`.¿{È|»wq.Kö~.áKR.ã!
000F:E880 0D7687A3 3AC18C39 D6828EC7 4F4C52B4 2581B3EE 1CC2C7C1 C761EC56 5C9BB1D0 .v.£:Á.9Ö..ÇOLR´%.³î.ÂÇÁÇaìV\.±Ð
000F:E8A0 A163 0A2E 00CDC0 85 4B48 05 6A E6 5E 7C 24 EA 88 AE ED 73 24 ECDB 0E 26 65 47 E3 11 33 13 ¡c..ÍÀ.KH.jæ^|$ê.®ís$ìÛ.&eGã.3.
```

e9a1ce3417838013412f81425ef74a37608754586722e00cacb333ba88eb9aa7.vhd-stage2

```
000F:F7A0 11 1B 2B 5D 4F 13 76 4B F6 95 9F 88 77 73 3C A2 70 43 8C 7A 81 2A FE E1 3E D5 A9 FC 4D EC 24 2A ..+]O.vKö...ws<¢pC.z.*þá>Õ©üMì$*
000F:F7C0 27 67 AA F3 B1 6A 92 B8 64 8E FE 08 74 C1 46 3C 54 6F 18 4C 10 D4 AA B8 E2 61 B7 04 27 39 5E 2E 'gªó±j.¸d.þ.tÁF<To.L.Ôª¸âa·.'9^.
000F:F7E0 49 0B 61 65 56 9A D7 12 5C 31 83 CD 94 77 82 ED 89 19 BF 31 A4 29 C4 44 BB 53 93 DC 36 0E 97 DA I.aeV.×.\1.Í.w.í..¿1¤)ÄD»S.Ü6..Ú
000F:F800 DE E6 FB D3 43 D4 3A C1 A7 82 4C F7 46 57 32 56 76 8E DD B2 9A CC CD E1 EE 40 2E 16 40 C1 82 D4 Þæ ûÓCÔ:Á§.L÷FW2Vv.Ý².ÌÍáî@..@Á.Ô
000F:F820 AB E0 FF 1C A3 70 50 AF 4B FA AE 31 96 D9 6D 8E 61 13 B1 10 AC FA EC 6B 6D C9 0C E3 3B 14 18 2E «àÿ.£pP¯Kú®1.Ùm.a.±.¬úìkmÉ.ã;...
000F:F840 06 84 90 2E F6 4A 2F 38 11 73 C7 4A FD 3C D4 E4 66 CA 60 8D 47 7D 0A 30 2F F7 AF E1 9A E7 96 07 ....öJ/8.sÇJý<ÔäfÊ`.G}.0/÷¯á.ç..
000F:F860 68 79 4B 55 4C 48 64 75 42 55 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 hyKULHduBU..........................
000F:F880 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 ................................
000F:F8A0 00000000 00000000 00000000 00000000 3CA6A26E E7C16CFD 19716F71 55B1854F ................<¦¢nçÁlý.qoqU±.O
```

Figure 5: Location of the blob and the RC4 key

The script below attempts to loop over data until a blob matches the blob size, then it extracts the RC4 key (the last 10 bytes of the blob) to finally decrypt the data.

```
 1
 2
 3
 4
 5
```

```
 from cryptography.hazmat.primitives.ciphers import Cipher
from cryptography.hazmat.primitives.ciphers.algorithms import ARC4


def decrypt_rc4(key: bytes, ciphertext: bytes) -> bytes:
```

```
    """Decrypt RC4 encrypt data, `pip install cryptography`"""

    algorithm = ARC4(key)
    cipher = Cipher(algorithm, mode=None)
    decryptor = cipher.decryptor()
    cleartext = decryptor.update(ciphertext)

    return cleartext


def get_bumblebee_c2(data: bytes) -> bytes:
    """
    Command and Control are stored at the end of the .data section,
    the configuration of the obfuscated C2 and its associated RC4
    are stored in the same blob with a fixed lenght of
    4105 plus one null byte (4106).
    !\xac\xd2\xfe=;\x87\x94\xebP\x8e@\x08}\x00/^I\xd4\x86\xaf\xd2\x14-
    \x16\x89A\xa9uT\x00\xbduC\xb7\x9e~\x19\xac\x9f\xb4\x0f\xae>\xcc
    \x96S]\xb56\x93C\x9d*p\xed\xc9\x04:Oew\xc3*X`:a\xe0T\x8e\x93>\xf9
    \xf8\xe2\x17Q\x15b,8\xa8[\xf5N\x93\xffMM]\x8d\xec\xde\x13\x95z\xc3
    ...
    ...
    ... <redatacted> ...
    \xd4\x00\xa1xZ:\x1e\x90\x00X\xea\xca\x0c\'\xee\xffOR5tw\xc0I\x86R"!
    \xf8\xa3\x87\xc8\x16Mo_5\x82_\x81\x9f<RC4 key composed by 10 bytes>
    """

    c2 = b""

    for blob in map(lambda x: x.strip(b"\x00"), data.split(b"\x00" * 4)):
        if len(blob) == 4106:
            key = blob[-10:]
            ciphertext = blob[:-10]
            c2 = decrypt_rc4(key, ciphertext)
            c2 = c2.replace(b"\x00", b"")
            print(f"BumbleBee Command and Control IoCs: {c2}")

    return c2


if __name__ == "__main__":
    import sys

    with open(sys.argv[1], "rb") as f:
        get_bumblebee_c2(f.read())
```

Code Snippet 1: BumbleBee C2 extractor

PS: Tested with the package `cryptography` *with the version:* `3.4.8`.

Go head and re-use, adapt the script for your needs!

Resources

- https://blog.google/threat-analysis-group/exposing-initial-access-broker-ties-conti/
- https://elis531989.medium.com/the-chronicles-of-bumblebee-the-hook-the-bee-and-the-trickbot-connection-686379311056
- https://malpedia.caad.fkie.fraunhofer.de/details/win.bumblebee
- https://0xtoxin-labs.gitbook.io/malware-analysis/malware-analysis/bumblebee-docusign-campaign

---