Bumblebee: increasing its capacity and evolving its TTPs

By itayc

Published: 2022-10-03 · Archived: 2026-04-26 02:14:41 UTC

Research by: Marc Salinas Fernandez

Background & Key Findings

The spring of 2022 saw a spike in activity of Bumblebee loader, a recent threat that has garnered a lot of attention due to its many links to several well-known malware families. In this piece we outline the conclusions of our research into this piece of malware:

- Bumblebee is in constant evolution, which is best demonstrated by the fact that the loader system has undergone a radical change twice in the range of a few days — first from the use of ISO format files to VHD format files containing a powershell script, then back again.
- Changes in the behavior of Bumblebee's servers that occurred around June 2022 indicate that the attackers may have shifted their focus from extensive testing of their malware to reach as many victims as possible.
- Although the threat contains a field called `group_name`, it may not be a good indicator for clustering-related activity: samples with different `group_name` values have been exhibiting similar behavior, which may indicate a single actor operating many `group_names`. The same is not true for encryption keys: different encryption keys generally imply different behavior, as expected.
- Bumblebee payloads vary greatly based on the type of victim. Infected standalone computers will likely be hit with banking trojans or infostealers, whereas organizational networks can expect to be hit with more advanced post-exploitation tools such as CobaltStrike.

Bumblebee Analysis

The Bumblebee loader usually comes in the form of a DLL-like binary packed with a custom packer. The method by which this DLL is delivered seems to be subject to change on the whims of the threat's adventurous developers: while the prevailing method is to embed the packed DLL directly inside another file (usually an ISO), during a short stint in June the malware's operators experimented with using VHD files that executed PowerShell downloading and decrypting the packed DLL itself (packed with a very different packer), as documented by Deep Instinct. This trend seems to have died out and now the DLL can be found directly embedded in the 1st-stage file again, whether an ISO or a VHD.

Once unpacked, Bumblebee will perform checks to avoid being executed in sandboxing or analyst environments; most of the code responsible for this is open source, lifted directly from the Al-Khaser project. If these checks pass, Bumblebee proceeds to load its configuration into memory. This is done by loading four pointers from its `.data` section which point to four different buffers in a contiguous encrypted configuration struct. The first of these points to an 80-byte section that stores an RC4 ascii key (much shorter in all cases we've observed). The other three pointers point to two 80-byte sections and a 1024-byte section, all of which contain data that is then decrypted using the above-mentioned RC4 key.

Once decrypted, the first 80-byte buffer in most of the samples to date has simply contained the number "444"; the malware makes no use of this number so its significance is not clear. The second buffer contains an ASCII code which is called `group_name` by the malware. Finally, the 1024-byte block contains a list of command and control servers (most of them are usually fake).

Figure 1: Bumblebee ciphered configuration

Bumblebee computes a machine-specific pseudorandom victim ID (internally named `client_id`) via the usual method of concatenating some immutable machine parameters (in this case, machine name and GUUID) and then calculating a hash of the result (in this case, an MD5 digest).

Using this data and some other elements collected from the victim system, Bumblebee builds a C&C check-in in JSON format, such as the one below:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

{

"client_id":"3f4aa6d4e02790dea90186c5376c0064",

"group_name":"1406r",

"sys_version":"Microsoft Windows 10 pro \\nUser name: LUCAS-PC\\nDomain name: WORKGROUP",

"client_version":1

}

{ "client_id":"3f4aa6d4e02790dea90186c5376c0064", "group_name":"1406r", "sys_version":"Microsoft Windows 10 pro \\nUser name: LUCAS-PC\\nDomain name: WORKGROUP", "client_version":1 }

```
{
  "client_id":"3f4aa6d4e02790dea90186c5376c0064",
  "group_name":"1406r",
  "sys_version":"Microsoft Windows 10 pro \\nUser name: LUCAS-PC\\nDomain name: WORKGROUP",
  "client_version":1
}
```

This string is encrypted using the same RC4 key used earlier for the configuration, and repeatedly sent to its C2 server with random delays between 25 seconds and 3 minutes regardless of whether the server responds or it's down. The response from the command and control server is also in JSON format and also encrypted with the same RC4 key (we appreciate this elegant design and encourage malware authors to aspire to this standard of legibility). The content of the response itself naturally varies, and can be for example an empty response:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

{

"response_status":1,

"tasks":null

}

{ "response_status":1, "tasks":null }

```
{
 "response_status":1,
 "tasks":null
}
```

Or some payload to inject or execute:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

{

"response_status": 1,

"tasks": [

{

"task": "shi",

"task_id": 5245,

"task_data":
"/EiD5PDowAAAAEFRQVBSUVZIMdJlSItSYEiLUhhIi1IgSItyUEgPt0pKTTHJSDHArDxhfAIsIEHByQ1BAcHi7VJBUUiLUiCLQjxIAdCLgIgAAAE

"file_entry_point": ""

}

]

}

{ "response_status": 1, "tasks": [{ "task": "shi", "task_id": 5245, "task_data":
"/EiD5PDowAAAAEFRQVBSUVZIMdJlSItSYEiLUhhIi1IgSItyUEgPt0pKTTHJSDHArDxhfAIsIEHByQ1BAcHi7VJBUUiLUiCLQjxIAdCLgIgAAAE
"file_entry_point": "" }] }


```
{
  "response_status": 1,
  "tasks": [
    {
      "task": "shi",
      "task_id": 5245,
      "task_data": "/EiD5PDowAAAAEFRQVBSUVZIMdJlSItSYEiLUhhIi1IgSItyUEgPt0pKTTHJSDHArDxhfAIsIEHByQ1BAcHi7VJBU
      "file_entry_point": ""
    }
  ]
}
```


In the case of receiving a payload, the structure of the response will contain a list of elements in the `tasks` section of the json, each with a command and a payload. Each of the elements will contain, among others, a `task` field with the name of the command to be executed, and a base64 encoded payload inside a section called `task_data`.

Botnet Behavior Analysis

Until early July we have observed a very curious behavior of the command and control servers. Once a `client_id` was generated for an infected victim and sent to a command and control server, that command and control server would stop accepting other different `client_id` codes from that same victim external IP. This means that if several computers in an organization, accessing the internet with the same public IP were infected, the C2 server will only accept the first one infected. But several weeks ago this feature was abruptly turned off, drastically increasing the number of established connections to infected victims at the expense of… whatever this feature was supposed to achieve (possibly it was indicative of a testing phase for the malware, which has now ended).

This behavior motivated us to pay special attention to the behavior of Bumblebee in different execution environments. Notably, despite having a field called `group_name` hardcoded in every sample, this value is sent in each request to the command and control server. Further, the above-described “one `client_id` per IP address” policy curiously seemed to apply across different `group_name`s — but not across different RC4 encryption keys, which seems to imply the use of several `group_name`s by what is effectively the same botnet, possibly to mark different campaigns or different sets of victims. As a result, grouping activity by encryption key seems to be a more coherent approach than grouping by `group_name`.

This hypothesis is further supported by the fact that we’ve observed several samples with the same RC4 key and different `group_name` acting identically and dropping the same threats within a very close time range, while samples that differ in their used RC4 key exhibit completely different behavior.

Figure 2: Different Bumblebee samples dropping the same payloads based on their RC4 Keys

The fact that command and control servers with different IP addresses contacted by different samples using the same RC4 key are returning the same payloads and blocking the same `client_id` for their victims also suggests that these IP addresses actually only act as fronts for a main command and control server to which all Bumblebee connections are relayed.

Another interesting element of the behavior of these botnets is how the toolset dropped by Bumblebee into victim machines differs depending on the kind of target. To deploy a threat, of the 5 commands supported by bumblebee, 3 lead to code being downloaded from the C2 server and executed:

- `DEX` : deploys an executable to disk and runs it.
- `DIJ` : Injects a library into a process and executes it.
- `SHI` : injects and executes shellcode into a process.

As part of our ongoing monitoring of various Bumblebee botnets, we have been monitoring differences in behavior based on factors such as type of network or geolocation. While the victim's geographical location didn't seem to have any effect on the malware behavior, we observed a very stark difference between the way Bumblebee behaves after infecting machines that are part of a domain (a logical group of network that share the same Active Directory server), as opposed to machines isolated from a company network that are connected to a workgroup (a Microsoft term to denote a peer to peer local area network).

If the victim is connected to WORKGROUP, in most cases it receives the `DEX` command (Download and Execute), which causes it to drop and run a file from the disk. These payloads are usually common stealers like Vidar Stealer, or banking trojans:

Figure 3: Bumblebee C2 response with a DEX command containing a Base64 encoded payload

On the other hand, if the victim is connected to an AD domain, it generally receives DIJ (Download and Inject) or SHI (Download shellcode and Inject) commands.

Figure 4: Bumblebee C2 response with a DIJ command containing a Base64 encoded payload

In these cases, the resulting threats have been payloads from more advanced post-exploitation frameworks, such as CobaltStrike, Sliver or Meterpreter.

In these cases, it has also been observed that regardless of the IP of the command and control server and the `group_name` field, samples with the same RC4 key drop the same Cobalt Strike beacons with the same Team servers, which has proven to be a very useful means of relating different samples to each other as part of the same botnet.

One last interesting feature of the payloads dropped by Bumblebee is that both the binaries downloaded using the DEX command and those downloaded with the DIJ command are in many cases packaged using the same Bumblebee packer.

Conclusion

Analyzing the behavior of the command and control servers used by Bumblebee operators, we have observed how they have tweaked the way their infection chains behave, sometimes in ways that served to drastically expand the number of active

victims and volume of C2 traffic.

For the moment, behavior until the deployment of the 2nd-stage payload is very similar even across different Bumblebee botnets, but further behavior starting with the choice of 2nd-stage payload sharply diverges based on RC4 key used. This behavior can also serve to group activity into different clusters, on top of using the RC4 key itself.

Unlike other threats that use third-party packers and off-the-crimeware-shelf antivirus evasion tools, Bumblebee uses its own packer both for the threat itself and for some of the samples it deploys on victims' computers, just like other advanced malware families such as Trickbot. While this allows Bumblebee operators greater flexibility in changing behavior and adding features, the use of unique custom tools also serves as a method to quickly identify Bumblebee activity in the wild.

Check Point's security products are designed to prevent any cyber attack and protect against threats such as described in this blog

Yara Rule

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

```
rule malware_bumblebee_packed {

meta:

author = "Marc Salinas @ CheckPoint Research"

malware_family = "BumbleBee"

date = "13/07/2022"

description = "Detects the packer used by bumblebee, the rule is based on the code responsible for allocating memory for a critical structure in its logic."

dll_jul = "6bc2ab410376c1587717b2293f2f3ce47cb341f4c527a729da28ce00adaaa8db"

dll_jun = "82aab01a3776e83695437f63dacda88a7e382af65af4af1306b5dbddbf34f9eb"

dll_may = "a5bcb48c0d29fbe956236107b074e66ffc61900bc5abfb127087bb1f4928615c"

iso_jul = "ca9da17b4b24bb5b24cc4274cc7040525092dffdaa5922f4a381e5e21ebf33aa"

iso_jun = "13c573cad2740d61e676440657b09033a5bec1e96aa1f404eed62ba819858d78"

iso_may = "b2c28cdc4468f65e6fe2f5ef3691fa682057ed51c4347ad6b9672a9e19b5565e"

zip_jun = "7024ec02c9670d02462764dcf99b9a66b29907eae5462edb7ae974fe2efeebad"

zip_may = "68ac44d1a9d77c25a97d2c443435459d757136f0d447bfe79027f7ef23a89fce"

strings:

$heapalloc = {

48 8? EC [1-6] // sub rsp, 80h

FF 15 ?? ?? 0? 00 [0-5] // call cs:GetProcessHeap

33 D2 // xor edx, edx ; dwFlags

4? [2-5] // mov rcx, rax ; hHeap

4? ?? ?? // mov r8d, ebx ; dwBytes

FF 15 ?? ?? 0? 00 // call cs:HeapAlloc

[8 - 11] // (load params)

48 89 05 ?? ?? ?? 00 // mov cs:HeapBufferPtr, rax

E8 ?? ?? ?? ?? // call memset

4? 8B ?? ?? ?? ?? 00 // mov r14, cs:HeapBufferPtr
```

}

condition:

$heapalloc

}

rule malware_bumblebee_packed { meta: author = "Marc Salinas @ CheckPoint Research" malware_family = "BumbleBee" date = "13/07/2022" description = "Detects the packer used by bumblebee, the rule is based on the code responsible for allocating memory for a critical structure in its logic." dll_jul = "6bc2ab410376c1587717b2293f2f3ce47cb341f4c527a729da28ce00adaaa8db" dll_jun = "82aab01a3776e83695437f63dacda88a7e382af65af4af1306b5dbddbf34f9eb" dll_may = "a5bcb48c0d29fbe956236107b074e66ffc61900bc5abfb127087bb1f4928615c" iso_jul = "ca9da17b4b24bb5b24cc4274cc7040525092dffdaa5922f4a381e5e21ebf33aa" iso_jun = "13c573cad2740d61e676440657b09033a5bec1e96aa1f404eed62ba819858d78" iso_may = "b2c28cdc4468f65e6fe2f5ef3691fa682057ed51c4347ad6b9672a9e19b5565e" zip_jun = "7024ec02c9670d02462764dcf99b9a66b29907eae5462edb7ae974fe2efeebad" zip_may = "68ac44d1a9d77c25a97d2c443435459d757136f0d447bfe79027f7ef23a89fce" strings: $heapalloc = { 48 8? EC [1-6] // sub rsp, 80h FF 15 ?? ?? 0? 00 [0-5] // call cs:GetProcessHeap 33 D2 // xor edx, edx ; dwFlags 4? [2-5] // mov rcx, rax ; hHeap 4? ?? ?? // mov r8d, ebx ; dwBytes FF 15 ?? ?? 0? 00 // call cs:HeapAlloc [8 - 11] // (load params) 48 89 05 ?? ?? ?? 00 // mov cs:HeapBufferPtr, rax E8 ?? ?? ?? ?? // call memset 4? 8B ?? ?? ?? ?? 00 // mov r14, cs:HeapBufferPtr } condition: $heapalloc }

```
rule malware_bumblebee_packed {
    meta:
        author = "Marc Salinas @ CheckPoint Research"
        malware_family = "BumbleBee"
        date = "13/07/2022"
        description = "Detects the packer used by bumblebee, the rule is based on the code responsible for al

        dll_jul = "6bc2ab410376c1587717b2293f2f3ce47cb341f4c527a729da28ce00adaaa8db"
        dll_jun = "82aab01a3776e83695437f63dacda88a7e382af65af4af1306b5dbddbf34f9eb"
        dll_may = "a5bcb48c0d29fbe956236107b074e66ffc61900bc5abfb127087bb1f4928615c"
        iso_jul = "ca9da17b4b24bb5b24cc4274cc7040525092dffdaa5922f4a381e5e21ebf33aa"
        iso_jun = "13c573cad2740d61e676440657b09033a5bec1e96aa1f404eed62ba819858d78"
        iso_may = "b2c28cdc4468f65e6fe2f5ef3691fa682057ed51c4347ad6b9672a9e19b5565e"
        zip_jun = "7024ec02c9670d02462764dcf99b9a66b29907eae5462edb7ae974fe2efeebad"
        zip_may = "68ac44d1a9d77c25a97d2c443435459d757136f0d447bfe79027f7ef23a89fce"

    strings:
        $heapalloc = {
            48 8? EC [1-6]           // sub     rsp, 80h
            FF 15 ?? ?? 0? 00 [0-5]  // call    cs:GetProcessHeap
            33 D2                    // xor     edx, edx        ; dwFlags
            4? [2-5]                 // mov     rcx, rax        ; hHeap
            4? ?? ??                 // mov     r8d, ebx        ; dwBytes
            FF 15 ?? ?? 0? 00        // call    cs:HeapAlloc
            [8 - 11]                 // (load params)
            48 89 05 ?? ?? ?? 00     // mov     cs:HeapBufferPtr, rax
            E8 ?? ?? ?? ??           // call    memset
            4? 8B ?? ?? ?? ?? 00     // mov     r14, cs:HeapBufferPtr
        }

    condition:
        $heapalloc
}
```

IOCs

Bumblebee samples

c70413851599bbcd9df3ce34cc356b66d10a5cbb2da97b488c1b68894c60ea69
14f04302df7fa49d138c876705303d6991083fd84c59e8a618d6933d50905c61
76e4742d9e7f4fd3a74a98c006dfdce23c2f9434e48809d62772acff169c3549
024f8b16ee749c7bb0d76500ab22aa1418cd8256fb12dcbf18ab248acf45947e

2691858396d4993749fec76ac34cf3cc3658ee3d4eaf9c748e2782cfc994849d
6bc2ab410376c1587717b2293f2f3ce47cb341f4c527a729da28ce00adaaa8db
083a4678c635f5d14ac5b6d15675d2b39f947bb9253be34d0ab0db18d3140f96
21df56d1d4b0a6a54bae3aba7fe15d307bac0e3391625cef9b05dd749cf78c0c
31005979dc726ed1ebfe05558f00c841912ca950dccdcdf73fd2ffbae1f2b97f
2d67a6e6e7f95d3649d4740419f596981a149b500503cbc3fcbeb11684e55218
3c0f67f71e427b24dc77b3dee60b08bfb19012634465115e1a2e7ee5bef16015
ca9da17b4b24bb5b24cc4274cc7040525092dffdaa5922f4a381e5e21ebf33aa
82aab01a3776e83695437f63dacda88a7e382af65af4af1306b5dbddbf34f9eb
a5bcb48c0d29fbe956236107b074e66ffc61900bc5abfb127087bb1f4928615c
07f277c527d707c6138aae2742939e8edc9f700e68c4f50fd3d17fe799641ea8
68ac44d1a9d77c25a97d2c443435459d757136f0d447bfe79027f7ef23a89fce
13c573cad2740d61e676440657b09033a5bec1e96aa1f404eed62ba819858d78
7024ec02c9670d02462764dcf99b9a66b29907eae5462edb7ae974fe2efeebad
ee27cceac88199bf3546e8b187d77509519d6782a0e114fc9cfc11faa2d33cd1
b2c28cdc4468f65e6fe2f5ef3691fa682057ed51c4347ad6b9672a9e19b5565e

Bumblebee C2 servers

104.168.201.219	142.11.234.230	145.239.30.26
145.239.135.155	145.239.28.110	146.19.173.202
146.70.125.122	152.89.247.79	185.17.40.189
185.62.58.175	205.185.122.143	205.185.123.137
209.141.46.50	209.141.58.141	51.210.158.156
51.68.144.94	51.68.145.54	51.68.146.186
51.68.147.233	51.75.62.99	51.83.250.240
51.83.251.245	51.83.253.131	51.83.253.244
54.37.130.166	54.37.131.14	54.38.136.111
54.38.136.187	54.38.138.94	54.38.139.20
