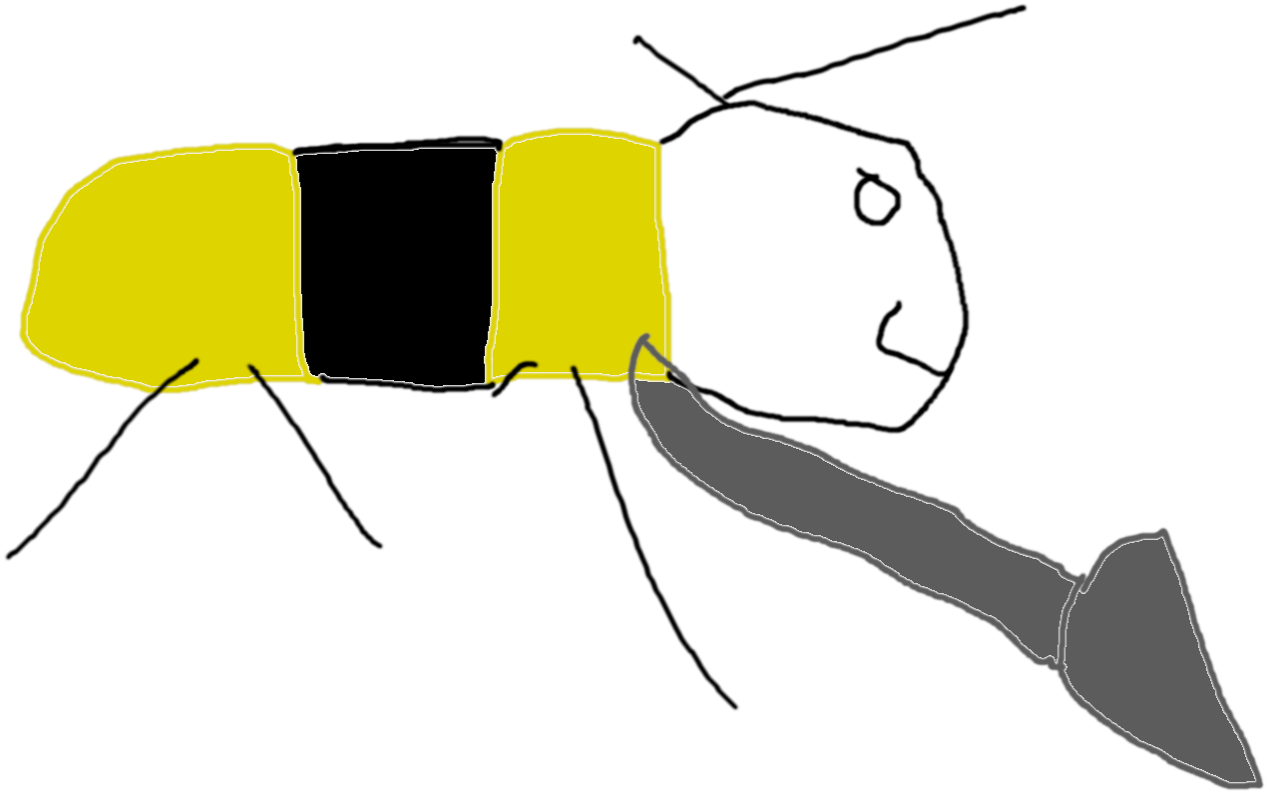
Bzz.. Bzz.. Bumblebee loader

threathunt.blog/bzz-bzz-bumblebee-loader

JouniMi May 8, 2022



Showing off my amazing graphical skills

Quite recently, a new loader has been popping up. This loader is likely been developed to counter the Microsoft's change to the macro behavior, as the macros will be disabled on the documents that have been downloaded from the internet. This is a very welcome change as macros have been often used by the threat actors to launch the malicious code from the maldocs. Now that this will be harder to the threat actors they are creating new creative ways to get the initial foothold after a phishing attack.

One of the new ways that has been observed in the wild is the bumblebee loader. This new loader will be delivered in format of an attachment, or a link within a phishing email. The user is then directed to a legitimate web service (for example, one drive) from which the user downloads a password-protected ZIP file, with the password in the email body. The zip contains an ISO file, which then contains two files – .lnk and .dat files. If the lnk file is started the bumblebee loader will be ran from the .dat file. There are several ways how this initial approach could be potentially found with Defender for Endpoint. To me it sounds like that the initial creation of the ISO file joined to the creation of the ZIP file by a browser could be a good approach. This could be maybe joined to network connections too, however it gets a little heavy and I leave it out for now.

So the logic of this query is explained above. I do use the time based join that I've used before on an earlier blog post. This is because the process that creates the ISO file is different from the process that created the archive file. Depending on the environment this could potentially need more joins to reduce noise. ISO files are relatively rare in some environments but much more common in the others. Also, this query is only looking for creations of the ISO + archive files – this does not yet mean that anything was executed.

```
let lookupWindow = 10min;
let lookupBin = lookupWindow / 2.0;
DeviceFileEvents
| where FileName endswith ".iso"
| where ActionType == 'FileCreated'
| extend TimeKey = bin(Timestamp, lookupBin)
| project DeviceName, IsoCreationTime = Timestamp, IsoCreationFileName = FileName,
IsoCreationFolderPath = FolderPath, IsoCreationSHA1 = SHA1, TimeKey,
IsoCreationProcessName = InitiatingProcessFileName, IsoCreationProcessCmdline =
InitiatingProcessCommandLine, IsoCreationProcessFolderPath =
InitiatingProcessFolderPath, IsoCreationParentName = InitiatingProcessParentFileName
| join (
DeviceFileEvents
| extend ArchiveCreationTime = Timestamp
| where FileName endswith ".zip" or FileName endswith ".rar" or FileName endswith
".7z"
| where InitiatingProcessFileName =~ "chrome.exe" or InitiatingProcessFileName =~
"firefox.exe" or InitiatingProcessFileName =~ "msedge.exe" or
InitiatingProcessFileName =~ "iexplore.exe"
| extend TimeKey = range(bin(Timestamp-lookupWindow, lookupBin), bin(Timestamp,
lookupBin), lookupBin)
| mv-expand TimeKey to typeof(datetime)
| project DeviceName, IsoCreationActionType= ActionType, ArchiveCreationTime =
Timestamp, ArchiveCreationFileName = FileName, ArchiveCreationFolderPath =
FolderPath, ArchiveCreationSHA1 = SHA1, TimeKey, ArchiveCreationProcessName =
InitiatingProcessFileName, ArchiveCreationProcessCmdline =
InitiatingProcessCommandLine, ArchiveCreationProcessFolderPath =
InitiatingProcessFolderPath, ArchiveCreationParentName =
InitiatingProcessParentFileName
) on DeviceName, TimeKey
| project DeviceName, IsoCreationTime, IsoCreationFileName, ArchiveCreationFileName,
IsoCreationProcessName, IsoCreationActionType, IsoCreationProcessCmdline,
IsoCreationProcessFolderPath, ArchiveCreationProcessName,
ArchiveCreationProcessCmdline, IsoCreationParentName, ArchiveCreationTime,
ArchiveCreationFolderPath, TimeKey, ArchiveCreationProcessFolderPath,
ArchiveCreationParentName, IsoCreationFolderPath, IsoCreationSHA1,
ArchiveCreationSHA1
```

As you can see from the query, I like to rename some of the fields. This does make it easier at least for me to understand the output especially when joining the data from multiple tables with different process names. I do this almost every time on my queries and I do also suggest it to others as well. The archive creation query is not limited to "FileCreated" events. This is because of how the modern browsers work, as they stage the files with different

names and then in the end renames the file to the final format. To test this out I created an empty file with the abbreviation of .ISO. Then I archived the file to a zip file and uploaded to Onedrive – from which I downloaded it and extracted the contents.

IsoCreationTime	IsoCreationFileName	ArchiveCreationFileName ↑	IsoCreationProcessName	IsoCreationActionType	IsoCreationProcessCmdline	IsoCreationProcessFolderPath	ArchiveCreationProcessName
May 8, 2022 9:23:49 AM	file.iso	badfile.zip	explorer.exe	FileRenamed	Explorer.EXE	c:\windows\explorer.exe	msedge.exe
May 8, 2022 9:23:49 AM	file.iso	badfile.zip	explorer.exe	FileRenamed	Explorer.EXE	c:\windows\explorer.exe	msedge.exe
May 8, 2022 9:23:49 AM	file.iso	badfile.zip	explorer.exe	FileModified	Explorer.EXE	c:\windows\explorer.exe	msedge.exe
May 8, 2022 9:23:49 AM	file.iso	badfile.zip	explorer.exe	FileModified	Explorer.EXE	c:\windows\explorer.exe	msedge.exe

The query matches the iso and archive creations.

The query works. There are multiple matches as there are multiple actions taken on the archive creation query. The default inner unique join takes one random entry from the left and joins it to all matches on right. This could be refined further but I am quite happy with this. One way to limit the results would be to limit the ActionType to “FileRenamed” on the archive creation query. The great article by Proofpoint (already linked before, but again here) states the process path after the malicious code is launched. It is basically cmd.exe -> cmd.exe -> rundll32.exe with certain switches. This should be easy enough.

```
DeviceProcessEvents
| where InitiatingProcessParentFileName =~ "cmd.exe"
| where InitiatingProcessFileName  =~ "cmd.exe"
| where InitiatingProcessCommandLine contains "IternalJob"
| where InitiatingProcessCommandLine contains "rundll32"
| where FileName =~ "rundll32.exe"
| where ProcessCommandLine contains "IternalJob"
| project Timestamp, DeviceName, InitiatingProcessParentFileName,
InitiatingProcessFileName, InitiatingProcessCommandLine, FileName, ProcessCommandLine
```

To keep the query efficient I didn’t do any joins. Unfortunately, the cmdline of the parent process is not recorded here so no filtering can be done based on that, unless joining the data. The query does not produce any results in my testing environment but your mileage may vary. I haven’t tested the queries in this post in any production environments yet.

These queries are relatively simple and likely will catch only partial and early versions of the loader, however I think they are a great start. Using these to start and then developing them a little further can produce nice results in catching the new type of loader. This data could of course also be joined further to get less results and to get more information out of the activity.

Also – someone might have noticed that I have changed the theme of the blog. I liked the old one but I migrated to another host as the first one was quite unresponsive at least from Europe. While doing that I changed the theme to handle couple of things better. I think there might be little bug here and there still with this but I will fix them when I get a chance.

EDIT: Fixed a typo in the latter query.