Malicious Image

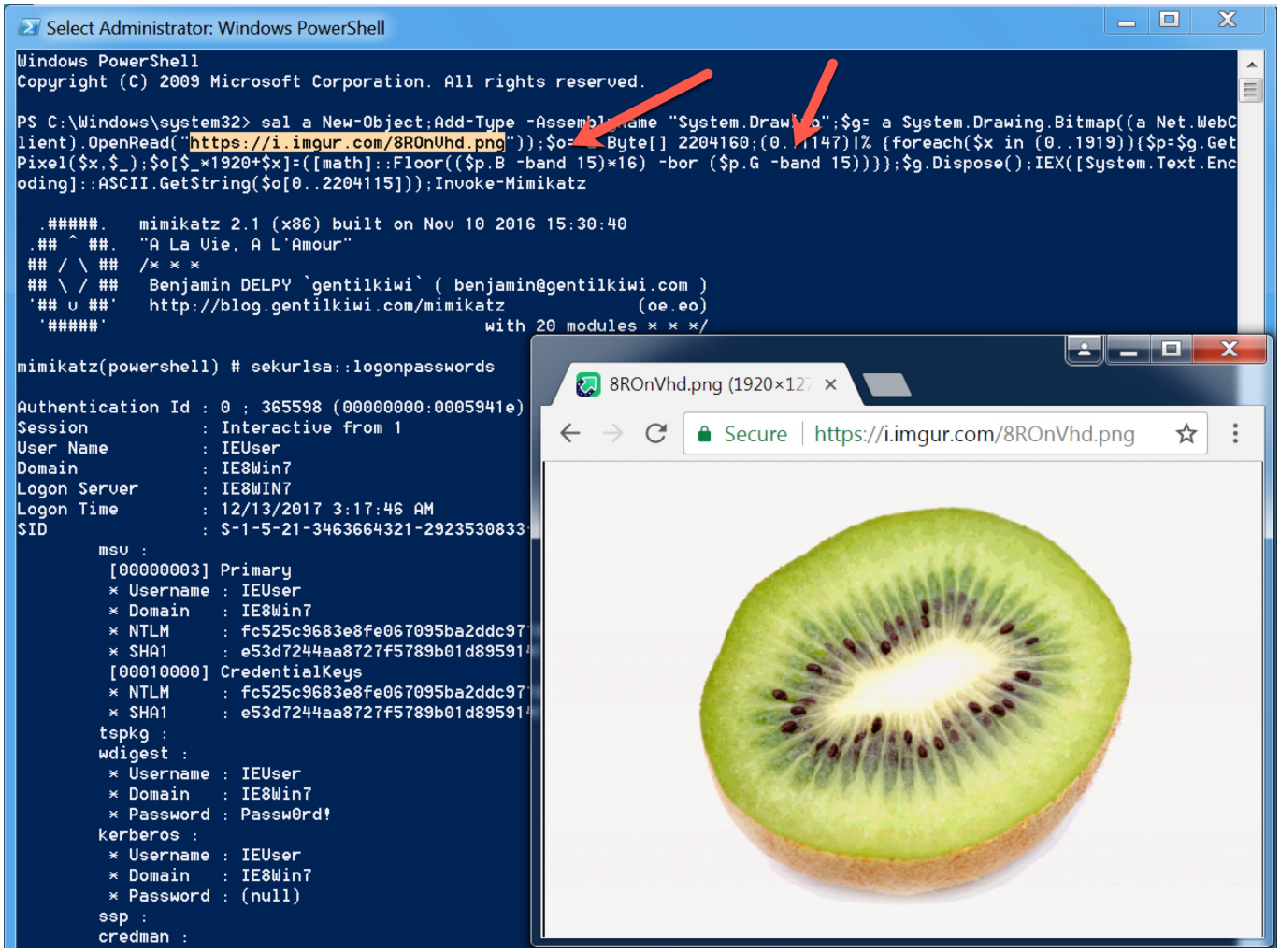
written by Mert SARICA | 1 December 2022

When we look at the campaigns carried out by APT groups such as Muddy Water, which also targets institutions in Turkey, we see that they sometimes use the Steganography technique. With this technique, cyber attackers try to infiltrate the target end user system through a social engineering attack and ensure that a malicious code fragment cannot be discerned visually and is downloaded and executed from the image file.

The first recorded uses of steganography can be traced back to 440 BC in Greece when Herodotus mentions two examples in his Histories. Histiaeus sent a message to his vassal, Aristagoras, by shaving the head of his most trusted servant, “marking” the message onto his scalp, then sending him on his way once his hair had regrown, with the instruction, “When thou art come to Miletus, bid Aristagoras shave thy head, and look thereon.” Additionally, Demaratus sent a warning about a forthcoming attack to Greece by writing it directly on the wooden backing of a wax tablet before applying its beeswax surface. Wax tablets were in common use then as reusable writing surfaces, sometimes used for shorthand.

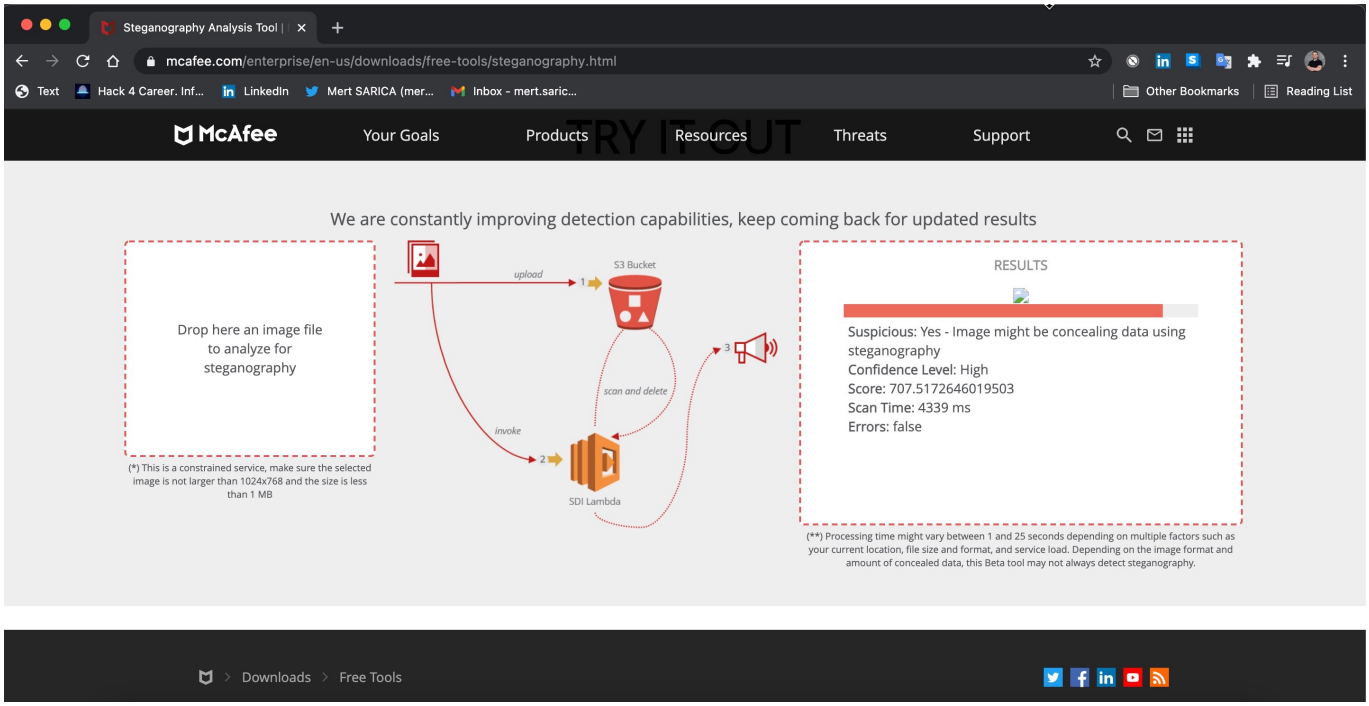
The Invoke-PSImage tool, which allows you to insert Powershell code into the pixels of a PNG image file, caught my attention after being featured in the news and on the Mitre T1027 technique page. After doing a Google search, I was unable to find a tool that could reveal the Powershell code in a PNG file created with this tool, so I decided to develop a tool that would be useful for incident response (IR) experts.

Initially, I looked at the source code to understand how the Invoke-PSImage tool works. I learned that it hides the Powershell code by manipulating the R (RED), G (GREEN), and B (BLUE) color codes (each of them is 8 bytes in size) of the target PNG file, specifically the least significant bits (LSB) of G and B, and by subjecting them to the same arithmetic operations ([math]::Floor((`$p.B-band15)*16)-bor(`$p.G -band 15)) each time (although this reduces the image quality, the human eye cannot easily detect the difference). Since it is practically possible to access the hidden Powershell code by reversing the operation that is constantly subjected to the same arithmetic operation, I was one step closer to my goal.

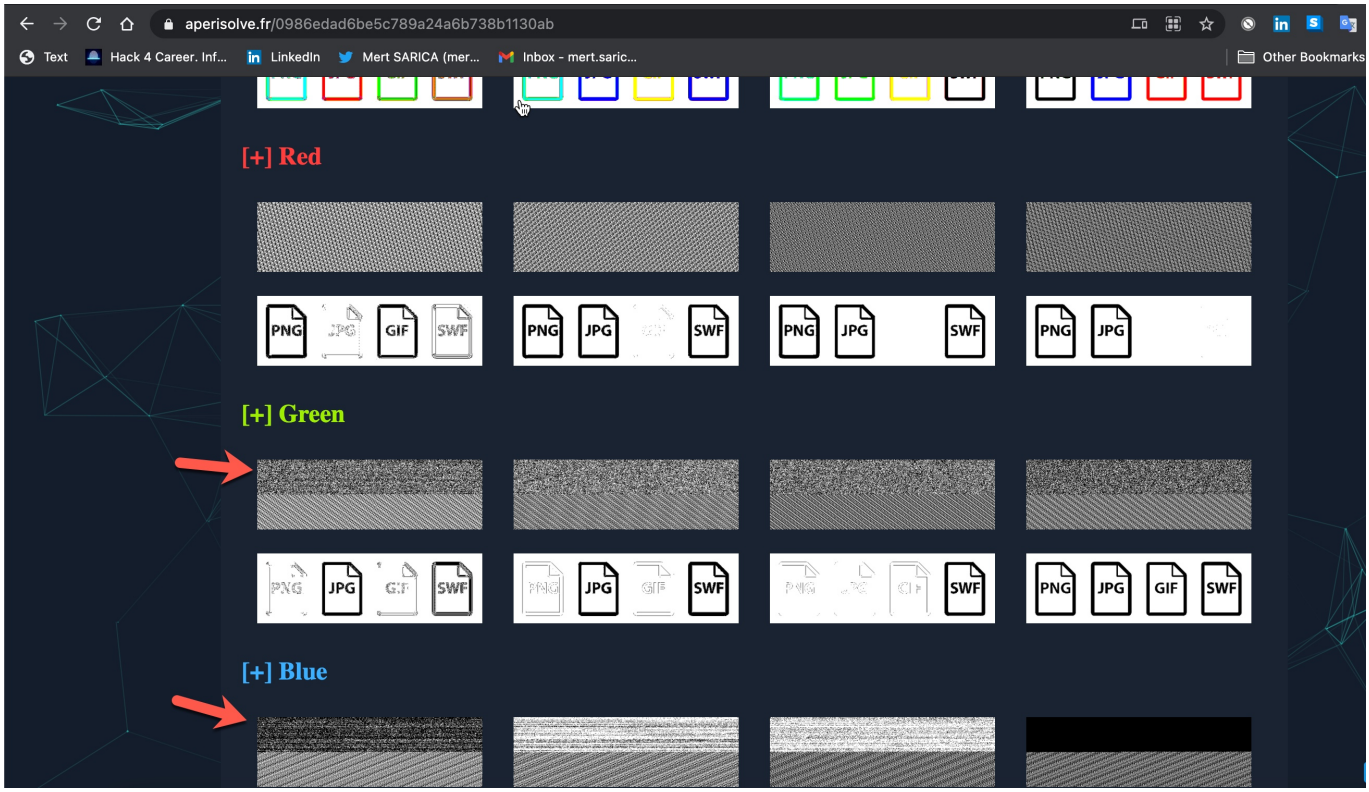


While searching for an easy way to determine whether there is a hidden message in the target image file before preparing the tool, I came across several tools and two handy websites. I got satisfactory results when I analyzed the file EPUWBt3.png used in a campaign by the Muddy Water APT group on both sites.

When I uploaded the file EPUWBt3.png to the first of these, the website of McAfee (FireEye – After merging with McAfee, they removed this tool from their website), I received a warning message that the file was suspicious.

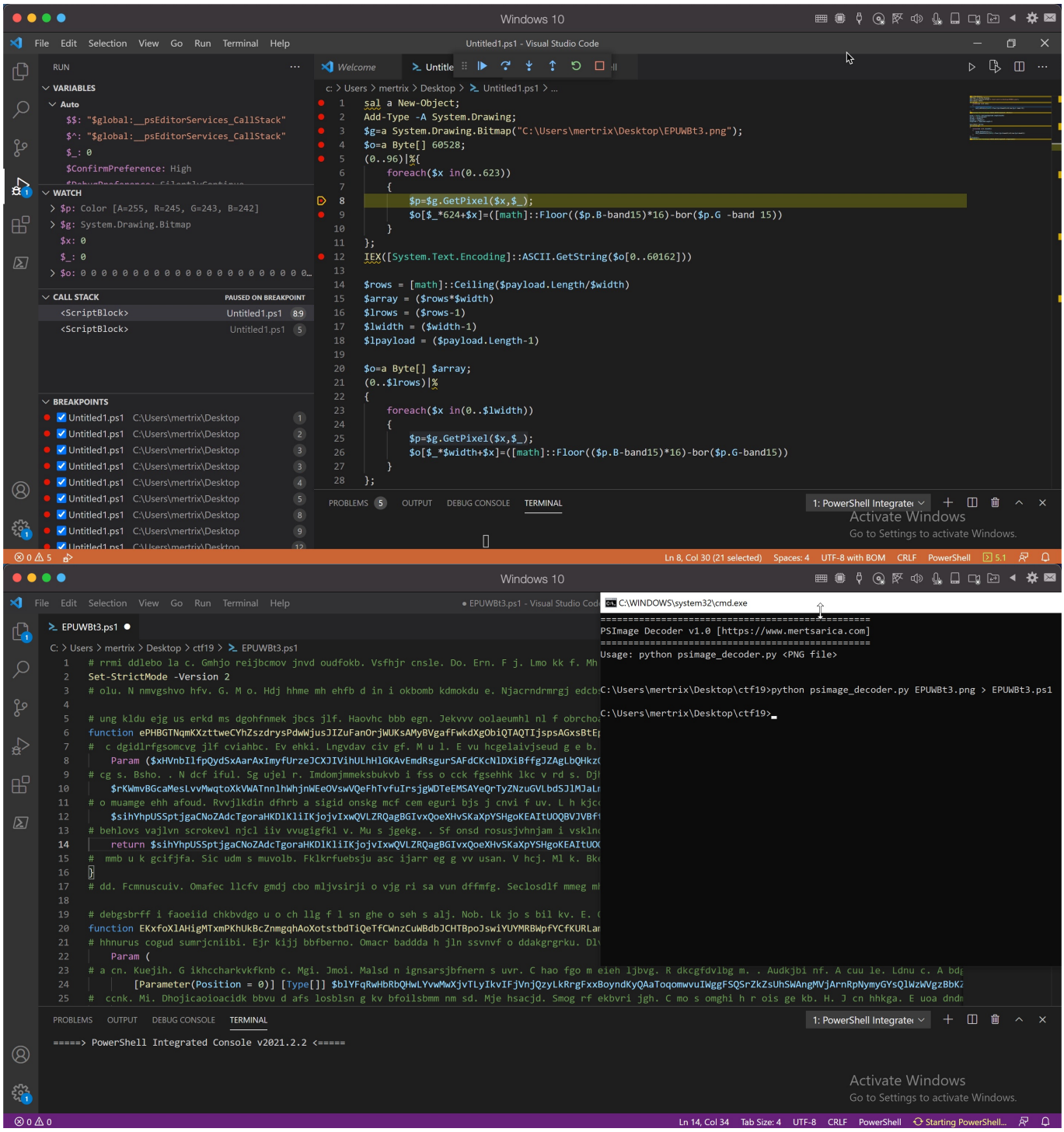


When I uploaded the file EPUWBt3.png to the other website called Aperi'Solve , the difference in the green and blue color codes, in particular, indicated that there was a suspicious situation in this image file.



When it was time to prepare the tool, after analyzing the Powershell code that deciphered the EPUWBt3.png image file step by step with Visual Studio Code using debugging, my new tool psimage_decoder.py, which reveals Powershell code hidden in image files using Invoke-PSImage, was ready for use

by cybersecurity experts.



Hope to see you in the following articles.

Note:

1. This article also contains the solution for the Pi Hediyem Var #19 cybersecurity game.