Practical 3

Antoine Ferguson
4/11/2022
a.ferguson@ufl.edu

Practical 3 - CAP4136

Executive Summary

I'm unsure what's going on with this malware, but I suspect code is being run from allocated heap memory. This is evident from the HeapCreate function that's imported and the string that states "Insufficient memory to perform operation.".

Static Analysis

PEStudio determined the malware was complied on March 4, 2022 at 12:51 UTC. Suspicious imports included PostThreadMessage, RegisterTypeLib, HeapCreate, IsDebuggerPresent, and GetCapture. Also when using pedump to explore the imported functions, I noticed all functions from the OLEAUT32.dll were imported as ordinals.

Some suspicious strings included, "Destination disk drive is full. Unable to read from %1.", "Command failed. Insufficient memory to perform operation.", and "Software\Microsoft\Windows\CurrentVersion\Policies\Network". I noticed some anti-disassembly techniques. For example, the instructions near 0x100010c1 had a return statement that would never execute because 0xC was added to the stack pointer, which skipped the instruction. Also various sections of code were misinterpreted by Ghirda as individual characters.

Dynamic Analysis

This DLL was forced to run using rundll32.dll with x32dbg. The sample3.dll file path and the ordinal of an exported functions were used as arguments with rundll32.dll. I couldn't verify any network connections, but I discovered 2 possible IPs (6.0.0.0 and 1.0.0.0) using a Yara rule. I also found the string, "Software\Microsoft\Windows\CurrentVersion\Policies\Network", which indicates that a network registry entry might've been used or created.

Indicators of Compromise

I had two Yara rules. The first rule confirmed the identity of the malware. I used the string "CustomMessageBox" to confirm the identify of the sample3.dll and I checked if it was a PE file. The full Yara rule was: 'strings: \n $m1 = "CustomMessageBox" \n condition: \n all of them and (uint16(0) == 0x5A4D and (uint32(uint32(0x3C)) == 0x00004550)'. My second rule checked for IP addresses that might be embedded in the DLL. The full Yara rule was: "strings: \n $ipv4 = /([0-9]{1,3}\.){3}[0-9]{1,3}/ wide ascii \n condition: \n $ipv4". My only indicator of infection was the rundll32 persisting after debugging the DLL with x32dgb.