**BLUE**

**Description:**

**Null**

**1. Initial Review using PESTUDIO**

Upon examining the indicator, I found that the sections involved self-modifying code with UPX0-2. UPX is a file compression tool used for executable files, often to reduce file size while maintaining the ability to execute. This suggests that the file had been compressed using UPX, which could potentially be a technique used to obfuscate the file's true content.

| indicator (24) | detail | level |
|---|---|---|
| virustotal > score | 16/71 | +++++ |
| sections > writable | UPX0 | +++++ |
| entry-point > location | 0x0003D2A0 | +++++ |
| sections > executable > count | 2 | +++++ |
| sections > self-modifying | UPX0 \| UPX1 | +++++ |
| sections > virtualized | UPX0 | ++ |
| sections > name > flag | UPX0 \| UPX1 \| UPX2 | ++ |
| imports > flag | 2 | ++ |
| file > entropy | 7.963 | + |
| file > type | executable | + |
| file > cpu | 64-bit | + |
| file > sha256 | 117A8AC437F0668661456AA4351D041056CCC1097D71D3E67CC1FF56F3... | + |

| general | | |
|---|---|---|
| subsystem | 0x0003 | **console** |
| magic | 0x020B | PE+ |
| file-checksum | 0x00000000 | 0x0002267E (expected) |
| entry-point | 0x0003D2A0 | section[UPX1] |
| base-of-code | 0x00027000 | section[UPX1] |
| size-of-code | 0x00018000 | 98304 bytes |
| size-of-initialized-data | 0x00001000 | 4096 bytes |
| size-of-uninitialized-data | 0x00026000 | 155648 bytes |

The entry point of the UPX1 compressed file was located at memory address 0x0003D2A0. This marks the starting point of the execution after UPX decompression, where the file begins its execution flow.

| property | value | value | value |
|---|---|---|---|
| section | section[0] | section[1] | section[2] |
| name | UPX0 | UPX1 | UPX2 |
| footprint > sha256 | n/a | F6AE46F2918E2DB52F135A4... | 96E9288D44C2D804DF196A... |
| entropy | n/a | 7.982 | 4.129 |
| file-ratio (99.47%) | n/a | 98.40 % | 1.06 % |
| raw-address (begin) | 0x00000200 | 0x00000200 | 0x00017400 |
| raw-address (end) | 0x00000200 | 0x00017400 | 0x00017800 |
| raw-size (95744 bytes) | 0x00000000 (0 bytes) | 0x00017200 (94720 bytes) | 0x00000400 (1024 bytes) |
| virtual-address | 0x00001000 | 0x00027000 | 0x0003F000 |
| virtual-size (258048 bytes) | 0x00026000 (155648 bytes) | 0x00018000 (98304 bytes) | 0x00001000 (4096 bytes) |
| | | | |
| characteristics | 0xE0000080 | 0xE0000040 | 0xC0000040 |
| write | x | x | x |
| execute | x | x | - |
| share | - | - | - |
| self-modifying | x | x | - |
| virtual | x | - | - |
| | | | |
| items | | | |
| directory > import | - | - | 0x0003F000 |
| directory > exception | - | 0x00039000 | - |
| directory > relocation | - | - | 0x0003F3C4 |
| directory > thread-local-storage | - | 0x0003DEC8 | - |
| directory > load-configuration | - | 0x0003DF58 | - |
| base-of-code | - | 0x00027000 | - |
| entry-point | - | 0x0003D2A0 | - |
| thread-local-storage | - | 0x0003DEA2 | - |

**UPX0:** This section supports write, execute, and self-modifying properties, indicating it can modify its own code during execution.

**UPX1:** This section allows write, execute, and self-modifying capabilities, but lacks the virtual properties seen in UPX0, limiting its self-modification scope.

**UPX2:** This section only allows write operations, without execution capabilities, suggesting it does not run code but can modify or overwrite data in the file.

### 2. Uncompressed using UPX

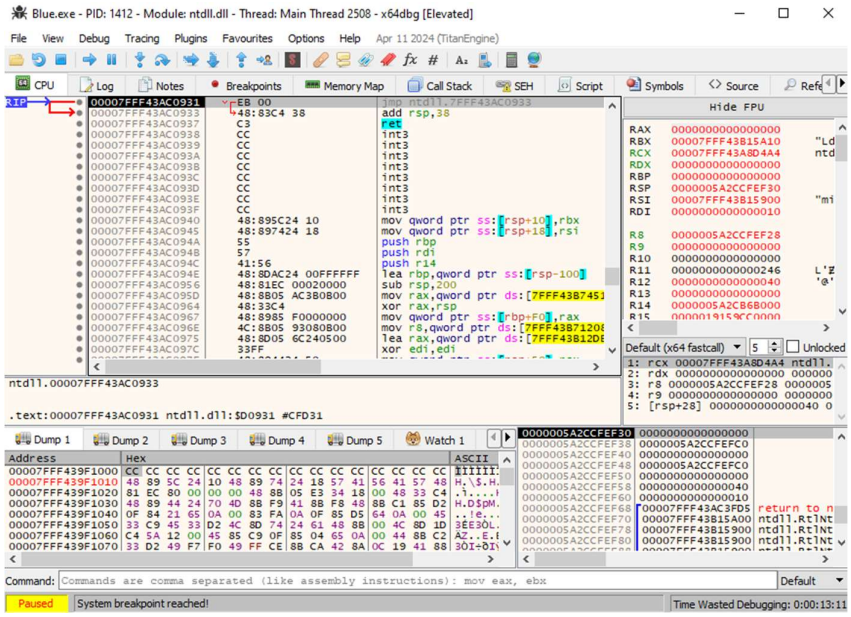
```
PS C:\Users\os1ris\Downloads\upx-4.2.4-win64 > .\upx.exe -d C:\Users\os1ris\Desktop\Blue.exe
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2024
UPX 4.2.4       Markus Oberhumer, Laszlo Molnar & John Reiser    May 9th 2024

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
    231424 <-     96256   41.59%    win64/pe     Blue.exe

Unpacked 1 file.
FLARE-VM 12/22/2024 05:33:36
```

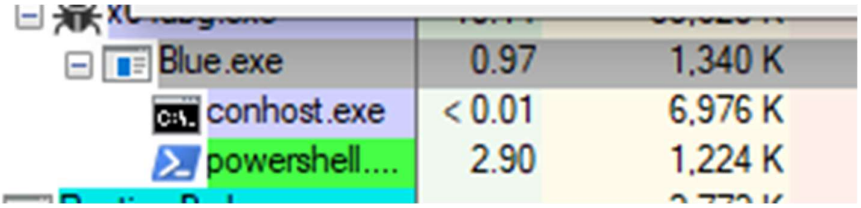
### 3. Reviewing the program using x64dbg

I then proceeded to decompress the UPX file using the UPX tool with the -d parameter. After performing static analysis, I found no plaintext or interesting information. Realizing that go through to the execution could be gained from the program's flows, I decided to conduct dynamic analysis to observe its behavior during runtime. **(Using my virtual machine for execution the malware)**

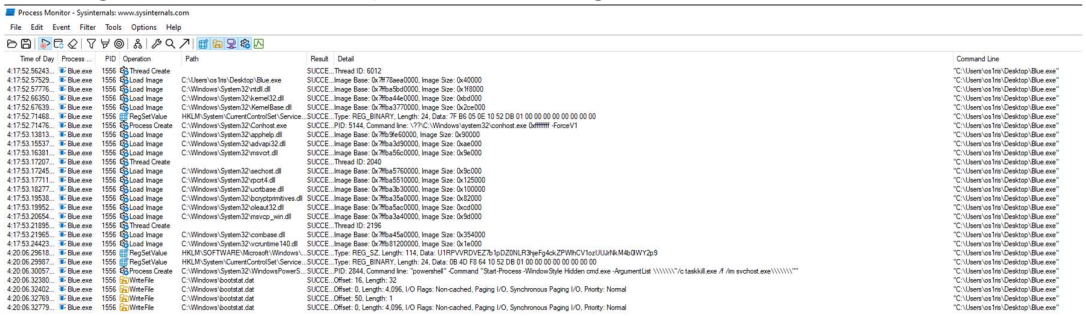



While attempting to analyze the program through debugging tools, I found it overwhelmed by the large amount of assembly code, making it difficult to go through each instruction one by one. Given the time constraints of the CTF, focusing on just one question felt like a significant effort. It's clearly that identifying the right spots to place breakpoints would take considerable time and patience. Interestingly, I can see some powershell execute on the process under the PID of **'Blue.exe'**

**4. Analysing the flow using Process Monitor**




I used Process Monitor to observe the activity of PowerShell and track the commands it was executing to trigger a BSOD on my PC. During this, I identified some encoded text starting with 'U1RPVVRD…', which I recognized as part of a flag format for 'STOUT'. This confirmed the presence of a flag within the encoded string, which I then saved into the RegSetValue (although we could stop here since we had already obtained the flag). However, I decided to continue my investigation to further explore the findings.



We can see to of the following, one of it was in Base64. It is a flag but I want to dig more.

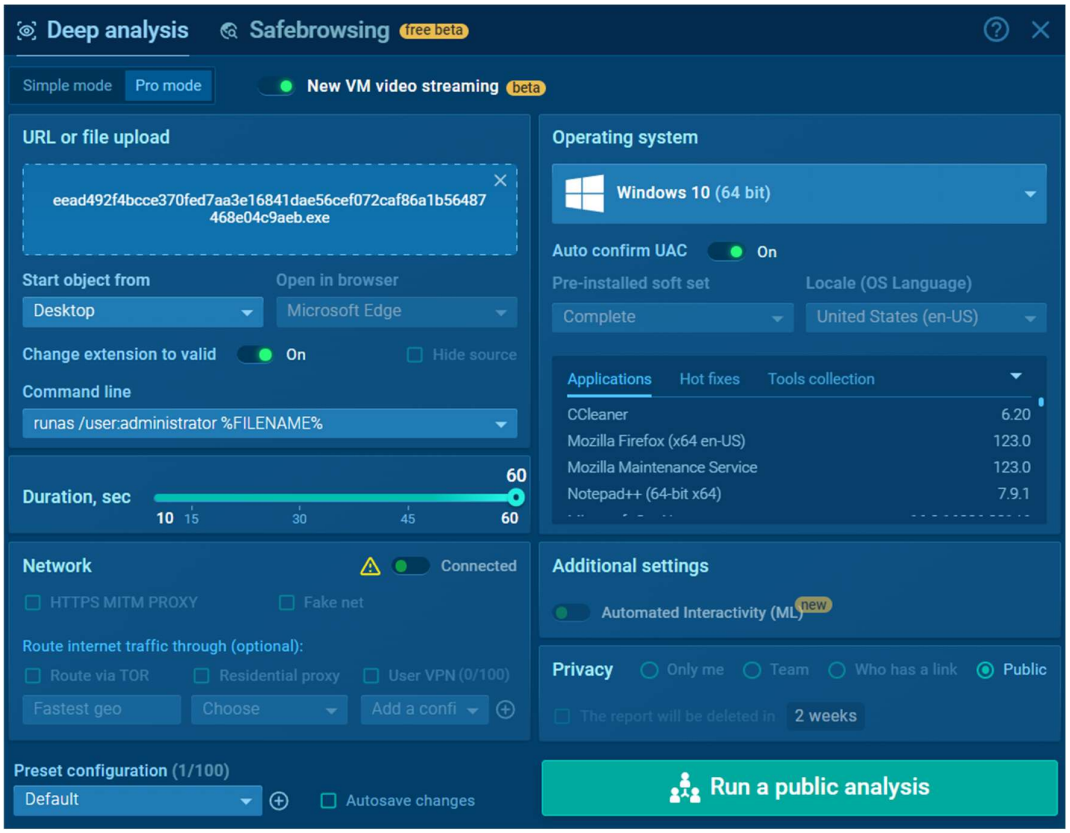
**IOC:**

```
U1RPVVRDVEZ7b1pDZ0NLR3hjeFg4ckZPWlhCV1ozUUJrNkM4b0lWY2p9
```

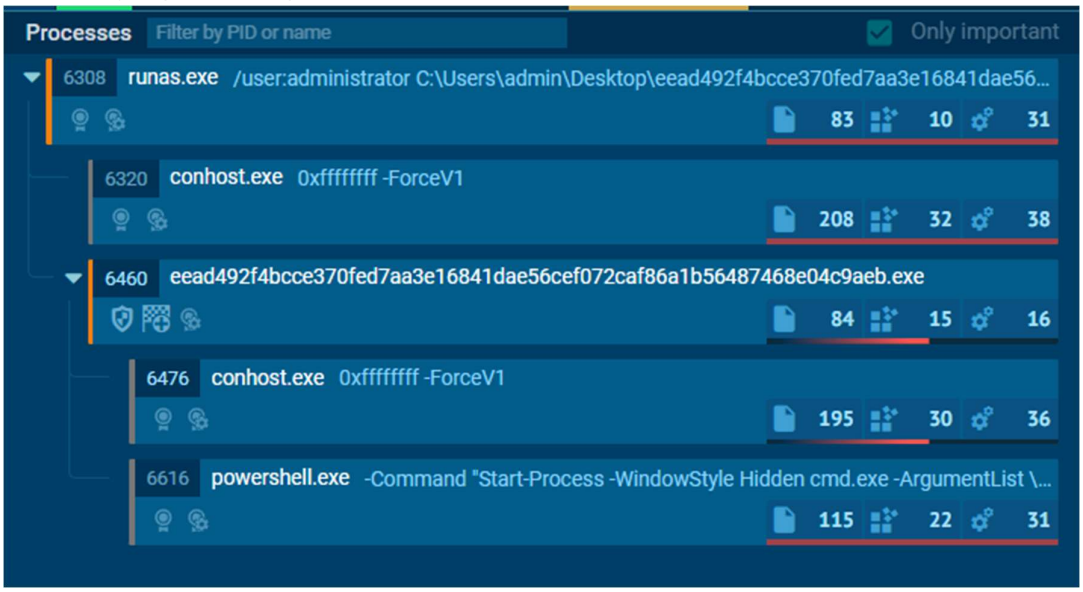
```
"POWERSHELL" -COMMAND "START-PROCESS -WINDOWSTYLE HIDDEN CMD.EXE -ARGUMENTLIST \\\\\\\"/C TASKKILL.EXE /F /IM SVCHOST.EXE\\\\\\\""
```

## ANALYSE WITH ANYRUN

### 1. Uploading into Any.Run



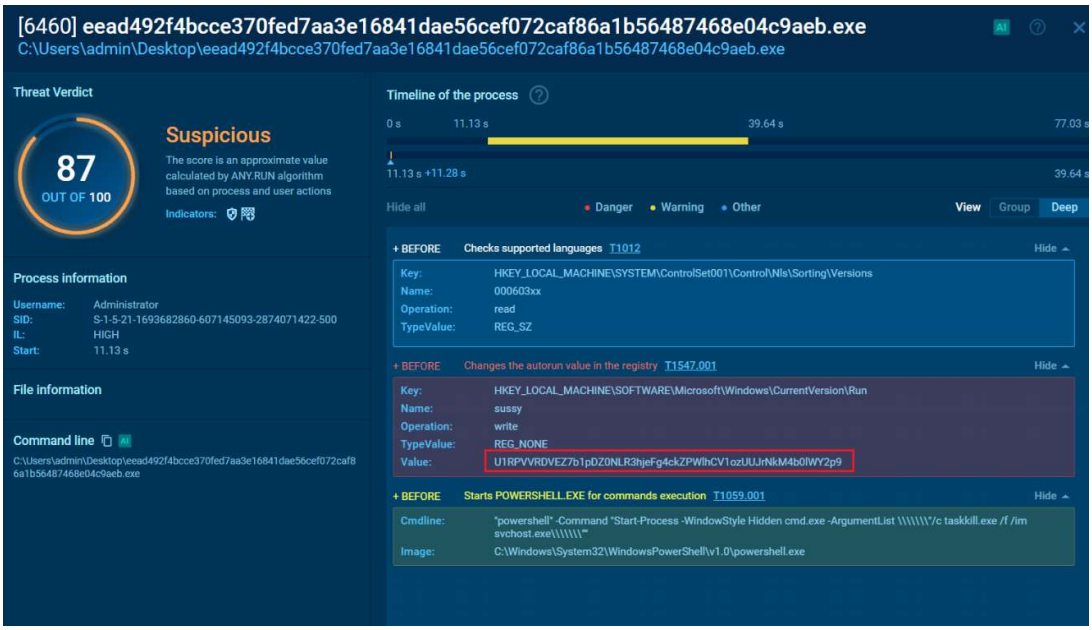

### 2. Verifying the program flow




Both ***conhost.exe*** and ***powershell.exe*** were running as child processes under the parent process of the executed application, ***Blue.exe***. I trying to check what does the ***Blue.exe*** does to execute the ***powershell*** and ***Registry***.

### 3. Clicking on the 'More Info'

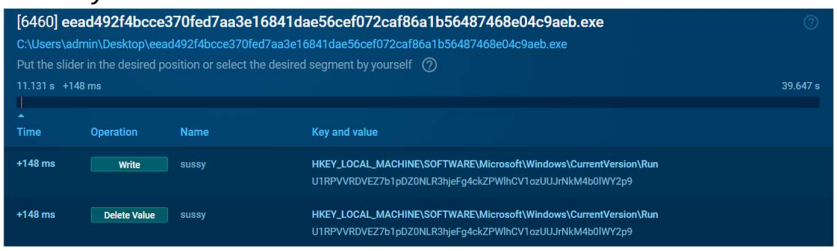
Clicking on the 'More Info' of the PID 6460 and found the registry set similar like the Process Monitor.




We can see the value was the flag stored inside the registry called ***sussy***. This key is used to store the startup programs for the Windows operating system. It is located in the registry and contains a list of programs that are **automatically executed when the system starts**. In this situation the program Write and Delete the value instantly inside:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
```

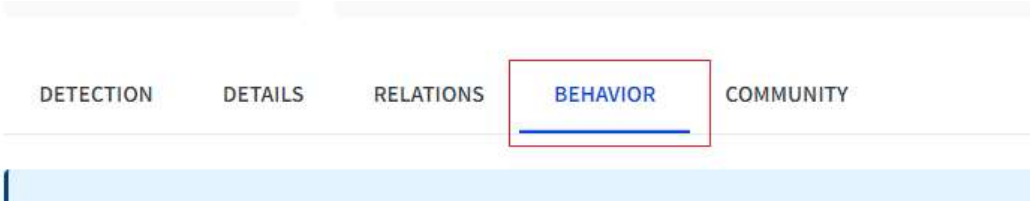
Malicious programs can also use this key to **add themselves to the startup programs** list, allowing them to execute without the user's knowledge or consent. This can be a technique used by malware to maintain persistence on the infected system and evade detection



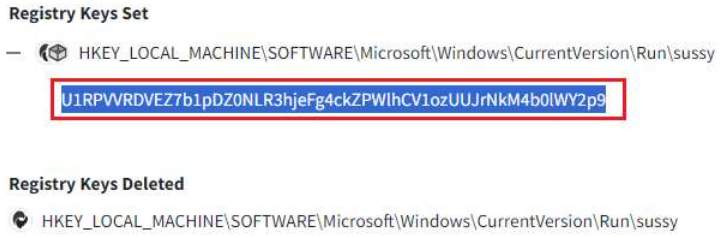

## UNINTENDED SOLUTION

1. **Uploading to Virustotal**


2. **Going into Behaviour:**




3. **Scroll and can see the Base64 encoded.**




| Flag | STOUTCTF{oZCgCKGxcxX8rFOZXBWZ3QBk6C8oIVcj} |
|---|---|