NYOTRON ATTACK RESPONSE CENTER

EyeD4kRAT/ShirBiter Overview

December 2020

Revision 0.2

NYOTRON

SECURING THE WORLD

EXECUTIVE SUMMARY

On November 30th, 2020, a **TLP:AMBER** document containing information about the **Shirbit cyberattack**, was uploaded to **TheMarker's** website, an Israeli business newspaper published by Haaretz group. We are not sure whether or not this was an accidental leak, but the fact that the document was marked with a limited disclosure restriction, definitely raises some questions.

While most of the data is publicly available to anyone without appropriate access to Shirbit's internal resources, **ClearSky Cyber Security** have had an interesting piece of information: a reference to a sample on VirusTotal that was allegedly used by the **Black Shadow** group after they've gained execution access to Shirbit's environment.

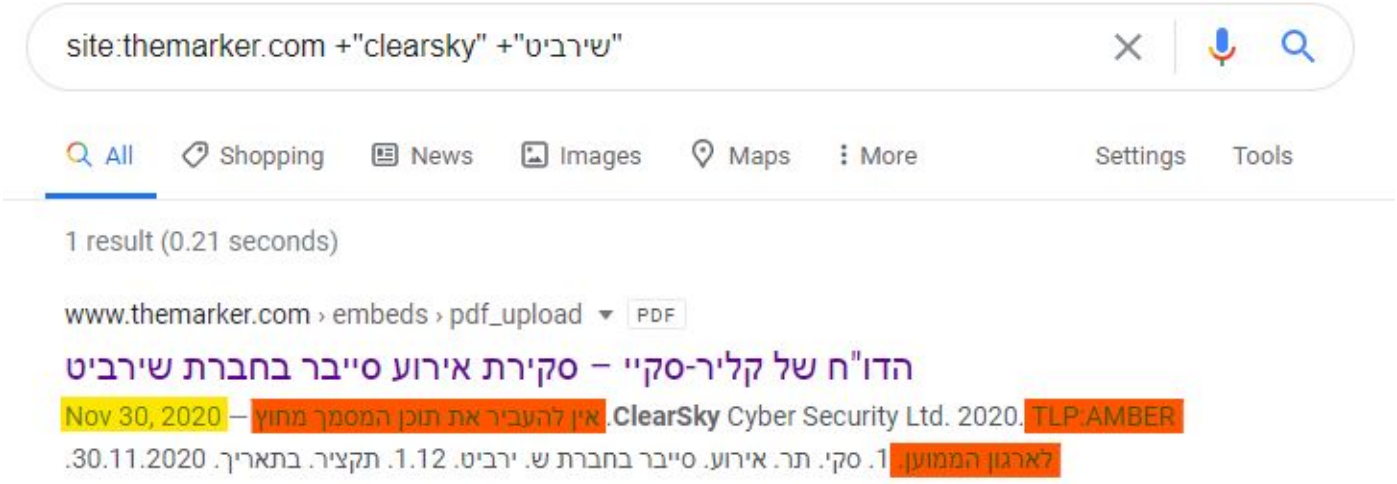
The purpose of this paper is to outline a high-level view of the sample's characteristics and capabilities, its appearances in the wild, and the stage where **Nyotron's PARANOID** has blocked it.

In this brief report, the threat will be referred to as ***"EyeD4kRAT"***. The name given is a combination of an embedded string used as the registry key name for storing entries, and a substring of its primary namespace ("new4k").

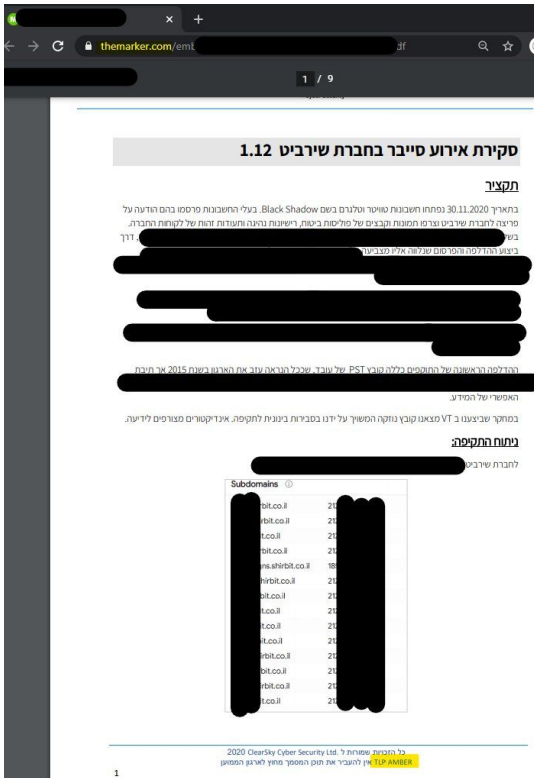
The sample in question (SHA256: 96cc69242a7900810c4d2e9f3f55aad8edb89137959f4c370f80a6e574ddc201) was neither highly sophisticated nor super-advanced in terms of stealthiness or anti-analysis. Rather, it was a non-obfuscated, very straightforward .NET assembly that has common trojan functionalities. That being said, its logic shows that the authors were likely experienced, and were probably following the *KISS* principle to make the tool look like a legitimate .NET program. The bottom line is that regardless of the attacker's skills - the malware got the job done pretty well for one side (unfortunately), despite its simplicity.

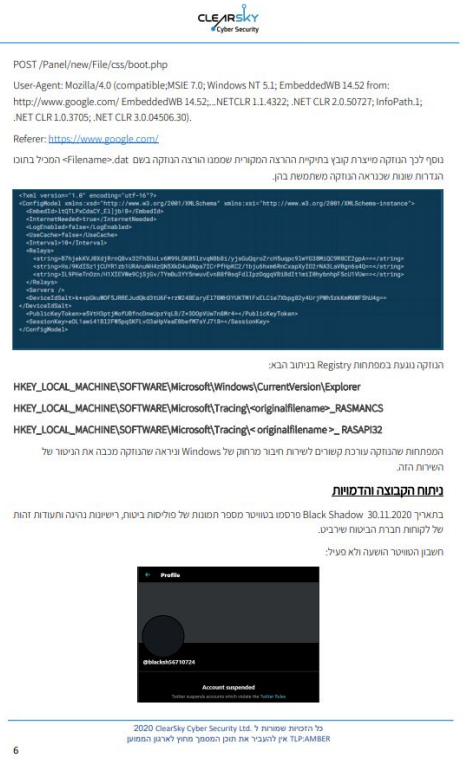
CLEARSKY'S REPORT

Screenshot of the Google search returning a **single search result**:



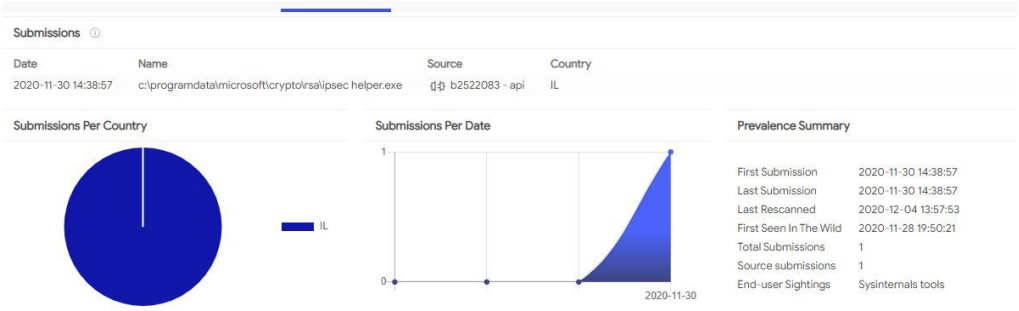
ClearSky's report screenshots (partially redacted):





VIRUSTOTAL DETECTIONS

EyeD4kRAT's sample that was mentioned by ClearSky was first uploaded to VirusTotal (VT) on November 30th 2020 via an Israeli host using VT-API, as shown in the screenshot below:



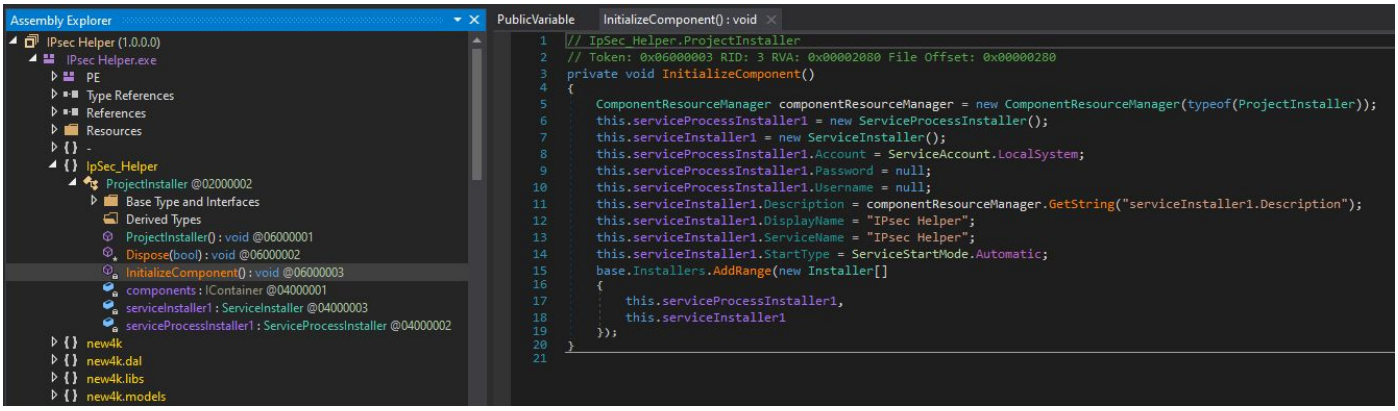
On the first analysis, VT's records shows that 7 out of 70 AV vendors detected the threat, as follows: **SecureAge APEX**, **BitDefenderTheta**, **Cynet**, **FireEye**, **Microsoft**, **Qihoo-360**, and **SentinelOne**. We can also see in the graph below detection rate evolution throughout the very first days after the breach was published:

Some of the responses returned by several engines since December 3rd were inconsistent, going back and forth from malicious to undetected and vice versa. Was the threat reclassified manually, or automatically? Well, it doesn't really matter. Either way, such errors could be dangerous for those who rely on that verdict. That is, despite the fact that people should generally avoid treating "undetected" as "benign".

EYED4KRAT HIGH LEVEL VIEW

Using dnSpy (or an equivalent) we can see EyeD4kRAT's namespaces:

1. **new4k** - Main program logic, command parsing loop
2. **IpSec_Helper** - Creates the local system service
3. **new4k.dal** - Data Access Layer, wraps filesystem, registry, and network APIs
4. **new4k.libs** - Commands functions' logic and hardcoded data
5. **new4k.models** - Data structures and configuration



The trojan uses plain HTTP POST requests for communicating with its C&C server(s), and contains the following major functionalities:

- Command line interface
- Download and execute
- Upload host files
- Gather host information
- Proxy support
- Persistence using SCM
- Leverages LOLBins
- Logging mechanism
- Update trojan engine
- Update trojan config
- Update C&C server(s)
- Self delete

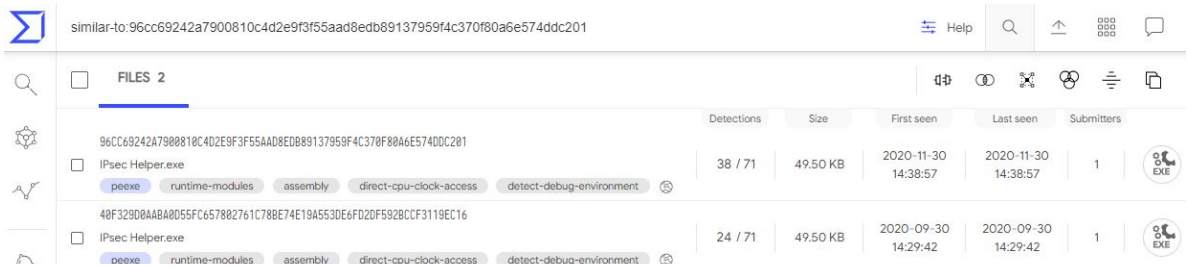
Although using an unencrypted protocol, the authors try to hide the transferred data and other saved information by encrypting it with the Rijndael algorithm, using the matching object as the key (e.g., config fields, logs, and payloads transferred from/to the server), and a hardcoded initialization vector that is widely used online in various public code examples. In addition to that, the actors hide the expected commands using hardcoded FNV-1a hashes.

Online references:

- https://gist.github.com/malkafly/00aae7f61a64c467d6e3#file-gistfile3-cs
- https://blog.xoc.net/2017/06/c-optimization-of-switch-statement-with.html

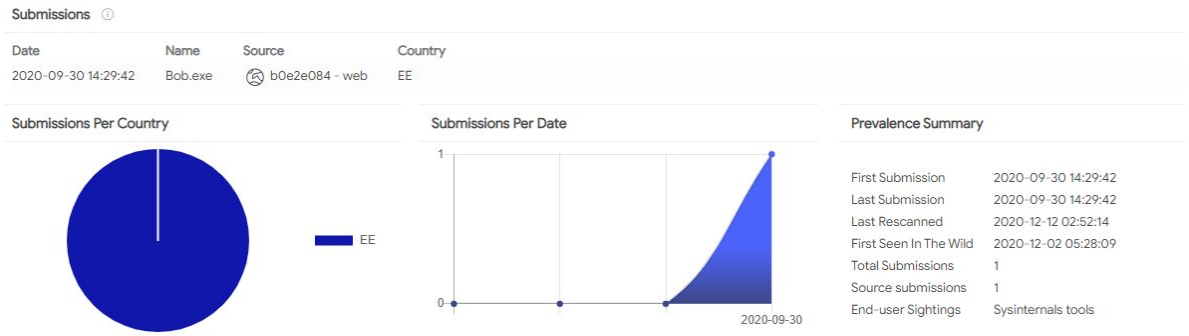
PREVIOUS APPEARANCES

With the great help of VirusTotal's *rich* features, we were able to find another sample that was first seen on VT two months earlier:



We believe that the older variant was uploaded by the authors, rather than an in-the-wild sample, since its embedded configuration had no C&C servers defined.

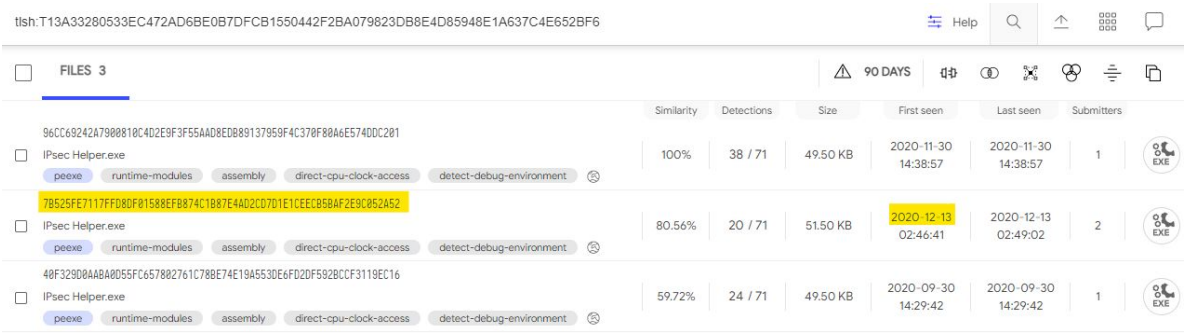
The sample (SHA256: `40f329d0aaba0d55fc657802761c78be74e19a553de6fd2df592bccf3119ec16`) was uploaded to VT via an Estonian host using the web interface (or an automated tool mimicking the same behavior? though using the API would be easier), and was seen with various filenames ("Bob.exe", "Service3.exe", and "IPsec Helper.exe").



While the upload of this lab-grown sample could have been a mistake the authors have made, it could also be intentional too. Maybe for checking the sample's detection rate, although that might seem quite odd as attackers usually use alternatives, as uploading to VT may get get their sample signed faster. The other speculation we can come up with would be an attempt to cause false attribution, but that's obviously a longshot. We're still trying to figure out how one could possibly upload such a file through VT's web interface by accident.

OTHER APPEARANCES

Just to mention its existence, we've seen an additional sample by searching for a similar Trend Micro Locality Sensitive Hash (TLSH), as well as the compilation time, which in this case is identical. Assuming the timestamp wasn't altered, it takes us back to Wednesday, July 29th of 2020, at 05:26:48 AM, UTC.

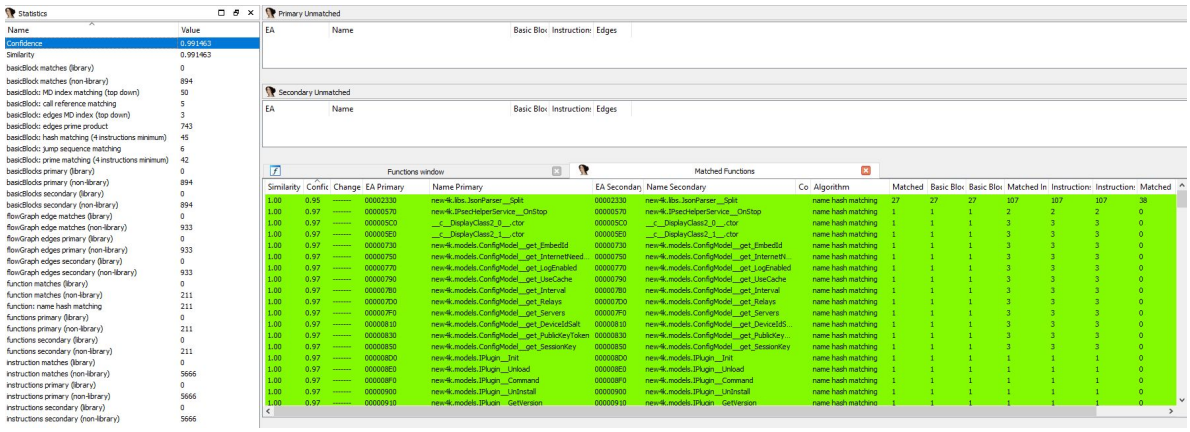


The sample (SHA256: 7b525fe7117ffd8df01588efb874c1b87e4ad2cd7d1e1ceecb5baf2e9c052a52) was uploaded twice using VT API. Comparing to the first upload, this seems like an opposite strategy (i.e., making it seem like it's automated), and this time, through China:

Since this sample was uploaded post to the aforementioned incident, it will not be referenced in this report again.

SPOT THE DIFFERENCE

By comparing both samples (hashes '96cc..' & '40f3..') with BinDiff on IDA, we can see that they are pretty much the same:



As previously mentioned, below we can see the '40f3' lab-sample config containing placeholders ('x.x.x.x') instead of real addresses, vs. the config of the '96cc' wild-sample, respectively:

```
public static string Data = "{\"EmbedId\":\"pdRTLJxDdaHU_Fljjc!Q\",\"InternetNeeded\":true,\"LogEnabled\":false,\"UseCache\":false,\"Interval\":10,\"Relays\":[\"http:\\/\\/x.x.x.x\\/
Panel\\/new\\/File\\/css\\/boot.php\",\"http:\\/\\/x.x.x.x\\/Scripts\\/_Data\\/25\\/lastupdate.php\",\"http:\\/\\/x.x.x.x\\/css\\/v1\\/template\\/main.php\"],\"DeviceIdSalt\":\"t
+tsGkuTOF5JRREJudQkd3tU6F+rzD43BEaryEl70WH3APRTM1FxELCie7Xbpg82y4UrjPWh5zkKmMXWF5hU4g==\",\"PublicKeyToken\":\"d1VtR9ptjMofUYfncEnwUpzYqLB\\/Y+3DOpVUw7n8Mr4=\",\"SessionKey\":
\"rOL1edi41B12FW5vfSKFLvO3aHpVeaE0befM7sYJ718=\",\"servers\":[]}";
```

```
public static string Data = "{\"EmbedId\":\"ltQTLPvCdaCY_El]jbIB\",\"InternetNeeded\":true,\"LogEnabled\":false,\"UseCache\":false,\"Interval\":10,\"Relays\":[\"http:\\/\\/5.2.73.67\
\/Panel\\/new\\/File\\/css\\/boot.php\",\"http:\\/\\/185.142.98.32\\/Scripts\\/_Data\\/25\\/lastupdate.php\",\"http:\\/\\/185.142.97.81\\/css\\/v1\\/template\\/main.php\"],
\"DeviceIdSalt\":\"k+xpGkuWOF5JRREJudQkd3tU6F+rzW24BEaryEl70WH3YUKTM1FxELCie7Xbpg82y4UrjPWh5zkKmMXWF5hU4g==\",\"PublicKeyToken\":\"e5VtH3ptjMofU8fncDnwUpzYqLB\\/Z+3DOpVUw7n8Mr4=\",
\"SessionKey\":\"eOL1awi41B12Fw5pqSKFLvO3aHpVeaE0befM7sYJ718=\",\"servers\":[]}";
```

Additionally, the service name of both samples is different, and it is unclear whether the attackers had a message they wanted to pass, or they were just trolling.

Operational sample

```
this.serviceInstaller1.DisplayName = "IPsec Helper";
this.serviceInstaller1.ServiceName = "IPsec Helper";
```

Lab sample

```
this.serviceInstaller1.DisplayName = "Bob Ross";
this.serviceInstaller1.ServiceName = "Bob Ross";
```

???

NYOTRON
SECURING THE WORLD

MORE HARDCODED VALUES

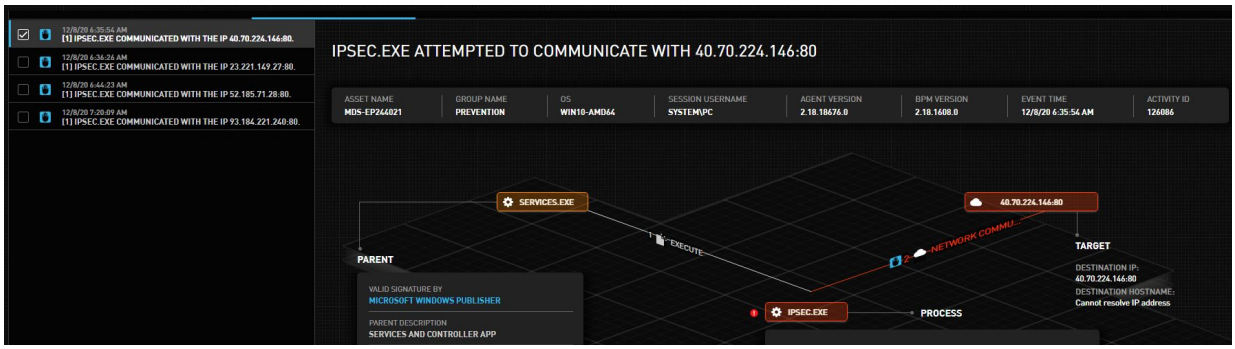
Both lab & wild EyeD4kRAT samples are of the same engine version, containing the list of hardcoded strings below, which are mostly self-explanatory (also, note the typo in the last variable's name):

PublicVariable

```
46      public static string EngineVersion = "2.15.5";
47
48      // Token: 0x04000051 RID: 81
49      public static string PowerShellRegistryPath = "HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\PowerShell\\1";
50
51      // Token: 0x04000052 RID: 82
52      public static string PowerShellRegistryName = "Install";
53
54      // Token: 0x04000053 RID: 83
55      public static string PowerShellRegistryValue = "1";
56
57      // Token: 0x04000054 RID: 84
58      public static string NodeInstallRegistryPath = "SOFTWARE\\Microsoft";
59
60      // Token: 0x04000055 RID: 85
61      public static string NodeInstallRegistryName = "Default";
62
63      // Token: 0x04000056 RID: 86
64      public static string NodeInstallRegistryValue = "140";
65
66      // Token: 0x04000057 RID: 87
67      public static string NodeInstallRegistryRunPath = "Software\\Microsoft\\Windows\\CurrentVersion\\Run";
68
69      // Token: 0x04000058 RID: 88
70      public static string NodeInstallRegistryRunName = "ipsecservice";
71
72      // Token: 0x04000059 RID: 89
73      public static string NodeInstallRegistryLocationPath = "Software\\Microsoft\\Windows\\CurrentVersion\\Explorer";
74
75      // Token: 0x0400005A RID: 90
76      public static string NodeInstallRegistryLocationName = "Signature";
77
78      // Token: 0x0400005B RID: 91
79      public static string NodeInstallRegistryConfigPath = "Software\\Microsoft\\Windows\\CurrentVersion\\Explorer";
80
81      // Token: 0x0400005C RID: 92
82      public static string NodeInstallRegistryConfigName = "Updater";
83
84      // Token: 0x0400005D RID: 93
85      public static string NodeIdRegistryPath = "Software\\Microsoft\\Windows\\CurrentVersion\\";
86
87      // Token: 0x0400005E RID: 94
88      public static string NodeIdRegistryName = "EyeD";
89
90      // Token: 0x0400005F RID: 95
91      public static string NodeConfigExtension = ".dat";
92
93      // Token: 0x04000060 RID: 96
94      public static string LogFileExtension = ".lgo";
95
96      // Token: 0x04000061 RID: 97
97      public static string CheckRelayParameter = "chk=Test";
98
99      // Token: 0x04000062 RID: 98
100     public static string CheckRelayResponse = "Ok";
101
102     // Token: 0x04000063 RID: 99
103     public static bool IsAlive = true;
104
105     // Token: 0x04000064 RID: 100
106     public static int CountOffRelay = 0;
107
```

PARANOID'S RESPONSE

Our team has executed the sample in our labs, and found that PARANOID prevented the communication attempts as expected:



NYOTRON
SECURING THE WORLD

ACKNOWLEDGEMENTS

Analysis & write-up by Freddy Ouzan.

Thanks to ClearSky & TheMarker for sharing the report.