# Nazar: A Lost Amulet

epicturla.com/blog/the-lost-nazar

April 22, 2020

Apr 22
Written By J A G-S

***Acknowledgements**: Special thanks to Silas Cutler for reversing guidance and to special friends (you know who you are) for visibility and insights.*

Accompanying talk presented on 04.22.2020 @ Virtual OPCDE #3 (Video)

Update #1 (04.22.2020) : Fixed some miscategorized hashes in the Appendix

Update #2 (04.23.2020): Reversing of EYService by @malwarelabpl available here.

Update #3 (04.28.2020): Additional reversing of the EYService comms protocol by @maciekkotowicz available here. I've added and corrected to reflect the new insights.

Update #4 (05.05.2020): Amazing in-depth analysis of Nazar components by Itay Cohen from Checkpoint available here.

---

Territorial Dispute continues to be an excellent resource for avid researchers undaunted by the thought of taking pointers from misplaced classified materials. For those blissfully unaware of *TeDi*, among the ShadowBrokers leaks we find two files far more noteworthy for threat intelligencers than the exploits and tools. Dr. Boldizar Bencsath and his team at CrySyS lab were the first to notice the value of *'sigs.py'* and '*drv_list.txt'.* The former includes filenames and registry keys associated umbrellaed under a moniker 'SIG[1-45]'. The CrySyS lab report is an excellent starting point to understand the contents of TeDi.

Since the release of this report in 2018, further signatures have been identified by Kaspersky's GReAT team and by Silas Cutler and I during our time at Chronicle's Uppercase. Today, I'll focus on a specific misidentified TeDi signature, *SIG37*. These signatures are low fidelity –composed of a combination of paths, filenames, and registry keys– and thereby prone to misidentification. In this case, CrySyS lab tentatively identifies SIG37 as 'IronTiger_ASPXSpy' –a presumably Chinese APT group better known as 'Emissary Panda' among other names. CrySyS lab points to a file in VirusTotal whose community comments suggest the aforementioned detection.

Venom23
2 years ago fb253831862d882b0d22cb2cb2a80d423cae92a6218ac3d126fafcadf75afd0b

Detected by THOR APT Scanner
Matched Rule: IronTiger_ASPXSpy
Ruleset: Iron Tiger

*Automated community comment with the misleading detection*

As many VT obsessives know, all sorts of misleading and undesirable files make their way into the VT corpus and fire off our YARA rules. Upon closer inspection, the file above is a 15mb memory dump of a McAfee installer, perhaps by someone interested in their malware signatures. As such, it's a giant malformed mess of unrelated indicators. With that understanding, we can discard the commented detection and return to the abject mystery of identifying SIG37. The signature itself refers only to a single filename: *'godown.dll'*

## The Nazar APT

```
def find_37():
    return ('godown.dll' in datastore.SYSTEMROOT_FILE_SET)
```

*SIG37 function from 'sigs.py'*

Armed with this spartan indicator, what we find is a previously unidentified cluster of activity possibly ranging as far back as early as 2008 –though more likely centered around 2010–2013 that I've nicknamed '*Nazar*'. The name is derived from debug paths left alongside Farsi resources in some of the malware droppers described below. Those PDB paths refer to a source root folder: '**khzer**'.

Contained Resources

| SHA-256 | File Type | Type | Language |
|---|---|---|---|
| 893cf8c164106784669b395825f17c21f46a345babfff6144686e8e1a48bf2f1 | ASCII text | REGISTRY | FARSI DEFAULT |
| 26ee0ff37e6ffd30ca5415992ececc5faeb8e6a937fcbeb3952ce5581456b7b5 | data | TYPELIB | FARSI DEFAULT |
| f1ebefcbe311522494a3654b10749d7e635cc3e4d052475ae4dd069486166597 | ASCII text | RT_STRING | ENGLISH US |
| 5488674c0ace4ab035ece1ecef07caedffb85e0f94a386362b791ce79e39a1d4 | data | RT_VERSION | ENGLISH US |

Debug Artifacts

Path C:\khzer\DLLs\DLL's Source\Filesystem\Debug\Filesystem.pdb

**Contained Resources**

| SHA-256 | File Type | Type | Language |
| --- | --- | --- | --- |
| 893cf8c164106784669b395825f17c21f46a345babfff6144686e8e1a48bf2f1 | ASCII text | REGISTRY | FARSI DEFAULT |
| 26ee0ff37e6ffd30ca5415992ececc5faeb8e6a937fcbeb3952ce5581456b7b5 | data | TYPELIB | FARSI DEFAULT |
| f1ebefcbe311522494a3654b10749d7e635cc3e4d052475ae4dd069486166597 | ASCII text | RT_STRING | ENGLISH US |
| 5488674c0ace4ab035ece1ecef07caedffb85e0f94a386362b791ce79e39a1d4 | data | RT_VERSION | ENGLISH US |



Native Farsi speakers pointed me in the direction of the term ‘nazar’ –roughly translating to ‘supervision’ or ‘monitoring’– transliterated and mangled from Persian to Roman characters. A more recognizable alternative interpretation is the nazar amulet used for protection against ‘evil eye’. Some level of speculation is involved so I won’t belabor the point beyond emojying the name ( 🧿) for tweetable convenience.

*‘Evil eye’ protection amulets– the better known ‘hamsa’ (left) or ‘nazar’ (right)*

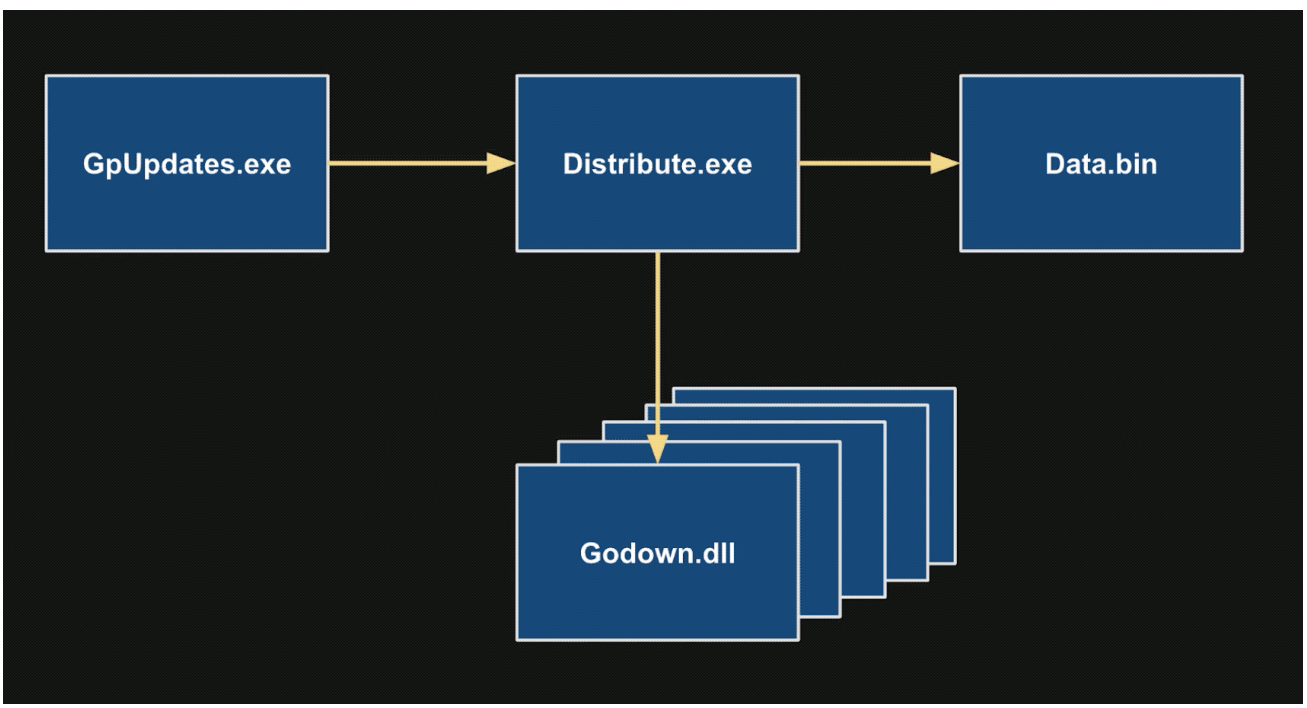
It’s hard to understand the scope of this operation without access to victimology (e.g.: endpoint visibility or command-and-control sinkholing). Additionally, some possible timestomping muddies the water between this operation possible originating in 2008-2009 or actually coming into full force in 2010-2013 (the latter dates being corroborated by VT firstseen submission times and second-stage drop timestamps). There’s a level of variable developmental capability visible throughout the stages. Multiple components are abused commonly-available resources, while the orchestrator and two of the DLL drops actually display some developmental ingenuity (in the form of seemingly novel COM techniques). Far from the most advanced coding practices but definitely better than the sort of .NET garbage other 'Farsi-speaking' APTs have gotten away with in the past.

Somehow, this operation found its way onto the NSA's radar pre-2013. As far as I can tell, it's eluded specific coverage from the security industry. A possible scenario to account for the disparate visibility between the NSA and Western researchers when it comes to this cluster of activity is that these samples were exclusively encountered on Iranian boxes overlapping with EQGRP implants. Submissions of Nazar subcomponents from Iran (as well as privately shared visibility into historical and ongoing victimology clustered entirely on Iranian machines) could support that theory. Perhaps this is an internal monitoring framework (*a la* Attor) but given the sparse availability of historical data, I wouldn't push that beyond a low-confidence assessment, at this time.

I hope interested researchers take this as an initial introduction and open challenge to contribute to what may prove a previously unknown threat actor, and encourage them to leverage their greater abilities and visibility to contribute to the ongoing research. I'll gladly update this post with the contributions and publications of others.

## Technical Breakdown

Nazar employs a modular toolkit where a main dropper silently registers multiple DLLs as OLE controls in the Windows registry via 'regsvr32.exe'. An orchestrator ('Data.bin'), disguised as the generic Windows service host process ('svchost.exe'), is registered as a service ('EYService') for persistence. The DLLs are a combination of custom type libraries and resourceful repurposing of more widely available libraries for nefarious purposes.




*Nazar component structure*

**The Dropper: 'GpUpdates.exe'**

| | |
|---|---|
| SHA256 | 4d0ab3951df93589a874192569cac88f7107f595600e274f52e2b75f68593bca |
| SHA1 | 48f99144bb9fdf379926e85fbe3caa462089f397 |
| MD5 | 6b3116580d29020b9c259877ac18a7fd |
| Filename | GpUpdates.exe |
| Timestamp | 2009-10-31 12:28:29 |
| First Submission | 2013-02-04 04:23:00 |

The droppers are misidentified as packed by Armadillo but in reality they're built using now defunct Chilkat software, 'Zip2Secure' to create self-extracting executables. The packing alone has led the droppers to be detected under generic AV detections but the subcomponents have low-to-no detections at this time.

**The Setup Agent: 'Distribute.exe'**

| | |
|---|---|
| SHA256 | 839c3e6ba65e5d07a2e0c4dd4a2c0d7ae95a266431dd3f8971b8a37d17b1ddf6 |
| SHA1 | 05122010cde4dcd1b4cd55de7b7d442efda19976 |
| MD5 | c1ab32afb0e2d7b7b1cad3fb831e9373 |
| Filename | Distribute.exe |
| Timestamp | 2012-03-17 11:07:53 |
| First Submission | 2013-02-04 04:26:29 |

The Zip2Secure configuration entrusts the distribution of the files contained therein to 'Distribute.exe', which places the files and silently registers the subcomponents with regsvr32.exe.

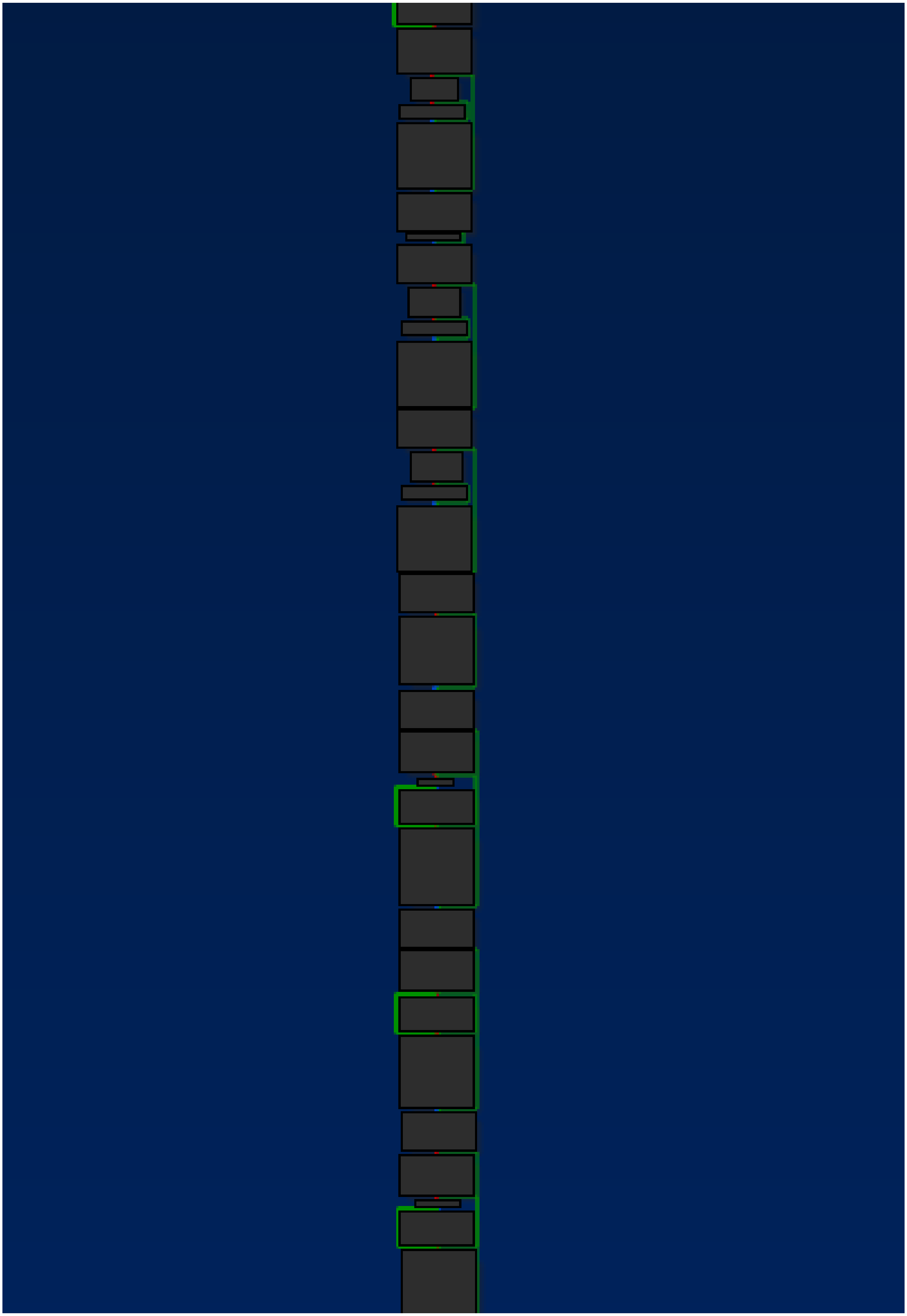
**The Orchestrator: 'EYService'**

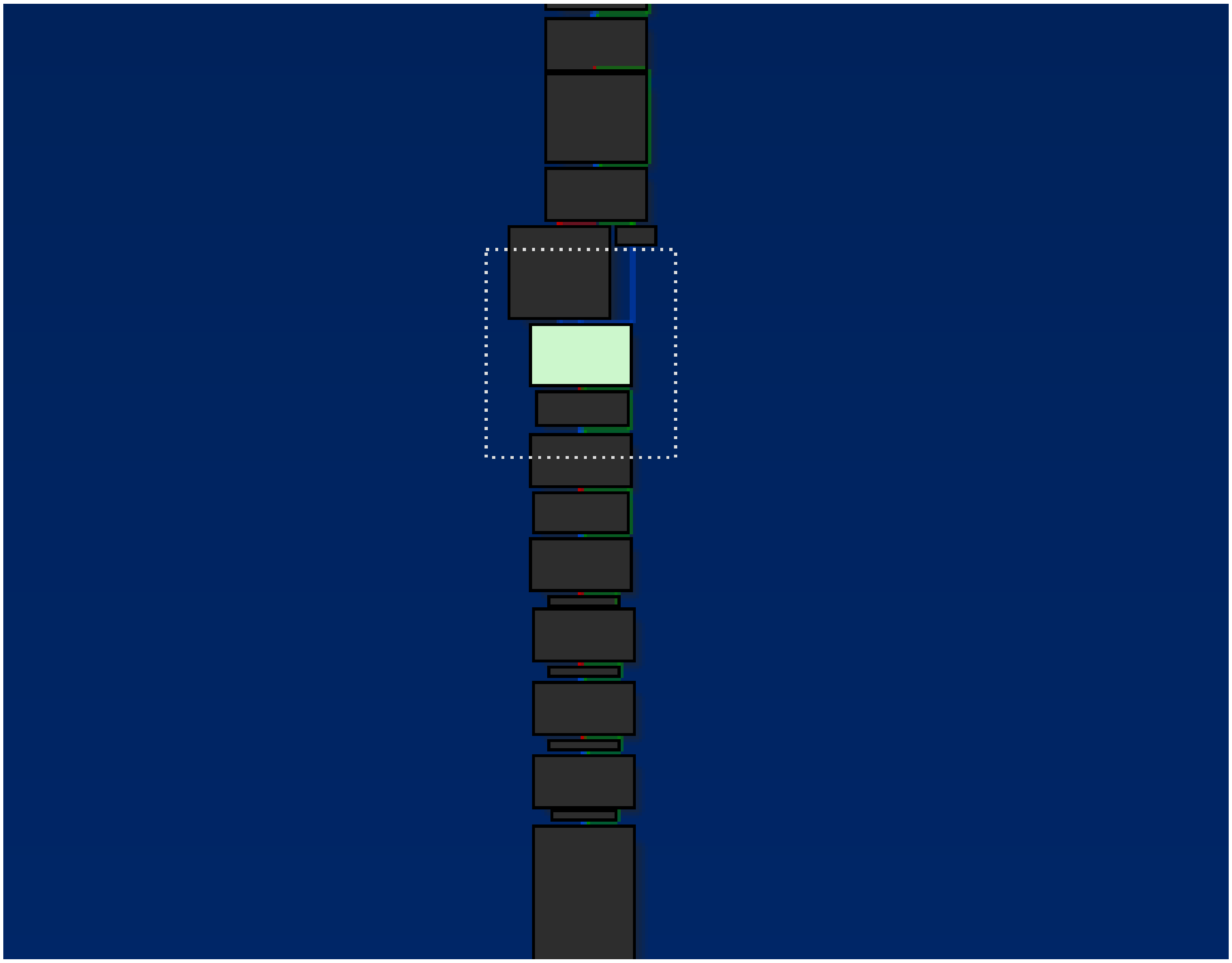
| | |
|---|---|
| SHA256 | 2fe9b76496a9480273357b6d35c012809bfa3ae8976813a7f5f4959402e3fbb6 |
| SHA1 | 79f2b98821c1e2717a0495e6c5c76a0147b21aae |
| MD5 | a9ff31c8db6d4e70829bf5db062d1b9c |
| Filename | Data.bin, svchost.exe, EYService |

| | |
|---|---|
| Timestamp | 2012-04-30 05:12:18 |
| First Submission | 2013-02-04 08:43:33 |

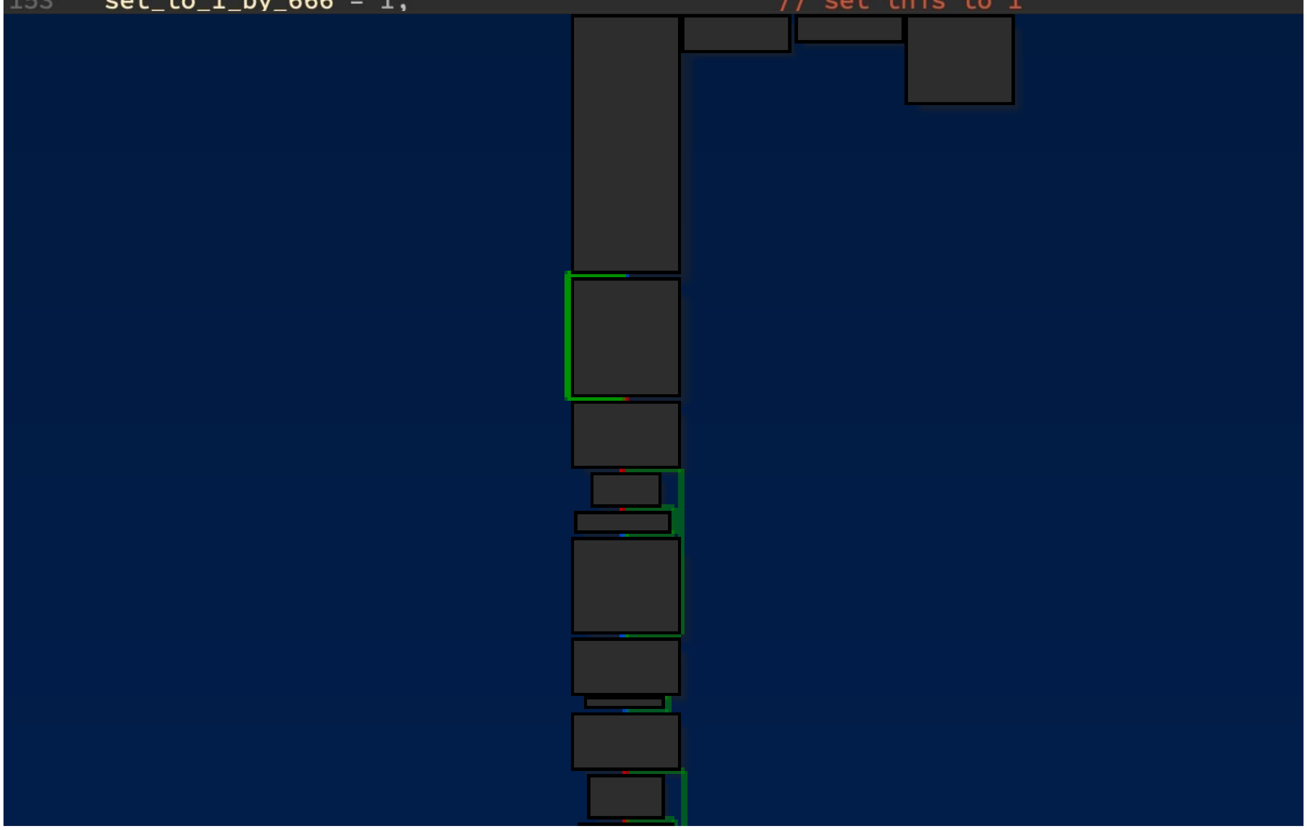
The main functionality orchestrating the different subcomponents is contained within Data.bin, later renamed to ‘svchost.exe’. The orchestrator takes 17 different three digit codes to divert functionality within a giant switch statement. Some of the codes have not been fully implemented up to the latest samples I’ve found so far, which further suggests a continued developmental effort.

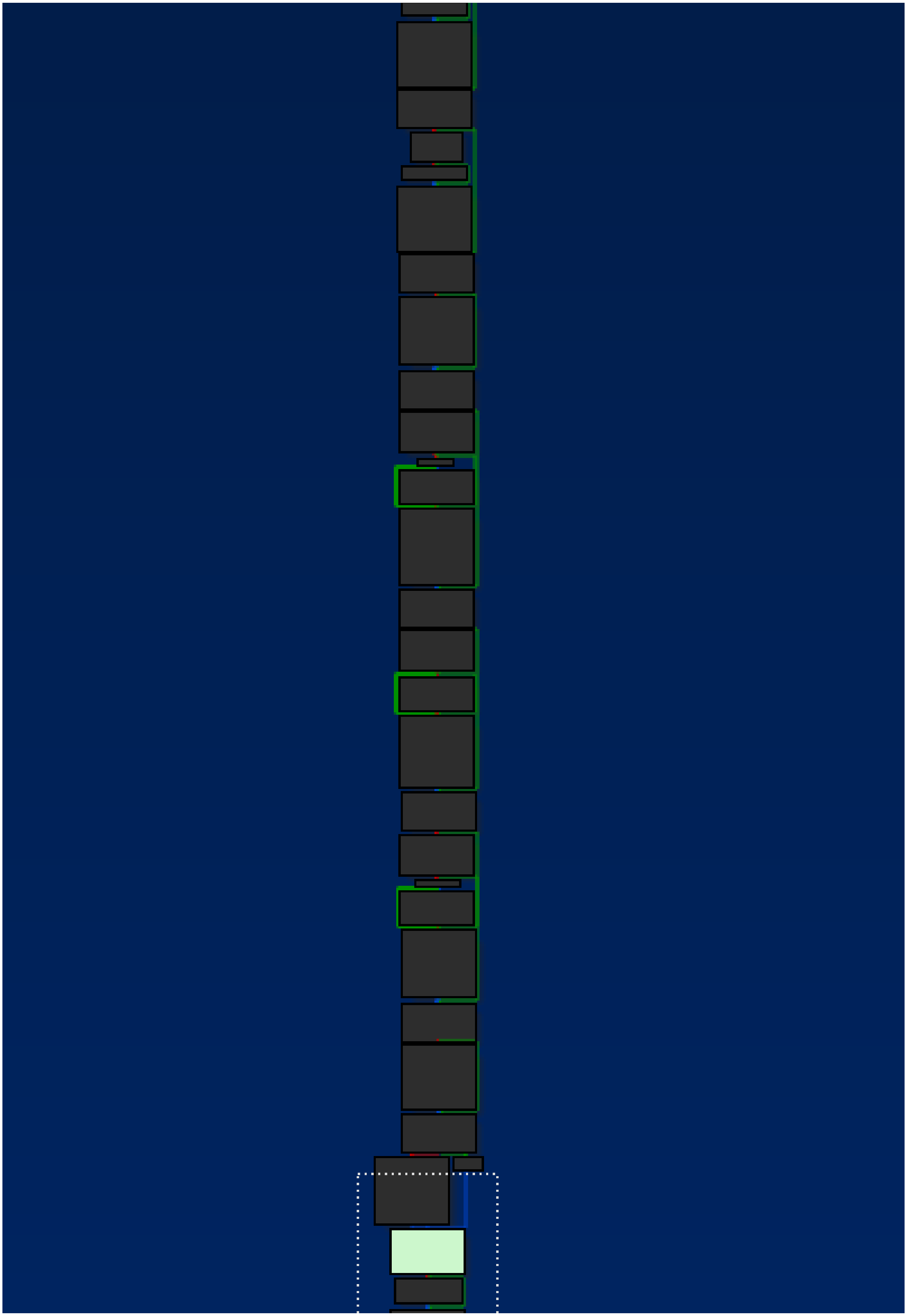
```
if ( (unsigned __int8)std::operator==<char>(switch_command, "499") )
{
  *(_DWORD *)&hostshort = unsigned4_1;
  hThread_4 = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)mw_writesCNCfromRegKey,
  ResumeThread(hThread_4);
  SetThreadPriority(hThread_4, 0);
  unsigned4_1 = unsigned4;
}
if ( (unsigned __int8)std::operator==<char>(switch_command, "599") )
{
  *(_DWORD *)&dword_436248 = unsigned4_1;
  ::hThread_2 = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)mw_599__, 0, 4u, &add
  ResumeThread(::hThread_2);
  SetThreadPriority(::hThread_2, 0);
}
if ( (unsigned __int8)std::operator==<char>(switch_command, "999") )// change C2
{
  *(_DWORD *)byte_4360E4 = 1;
  strncpy(cp, Source, 0x10u);
}
if ( (unsigned __int8)std::operator==<char>(switch_command, "555") )
{
  strncpy(cp, Source, 0x10u);
  mw_THIS_is_where_C2_is_defined(1);
}
if ( (unsigned __int8)std::operator==<char>(switch_command, "315") )
  set_to_0_by_315 = 0;                         // zero this out
if ( (unsigned __int8)std::operator==<char>(switch_command, "312") )
  set_to_0_by_312 = 0;                         // zero this out
if ( (unsigned __int8)std::operator==<char>(switch_command, "313") )
  set_to_0_by_313 = 0;                         // zero this out
if ( (unsigned __int8)std::operator==<char>(switch_command, "666") )
  set_to_1_by_666 = 1;                         // set this to 1
```

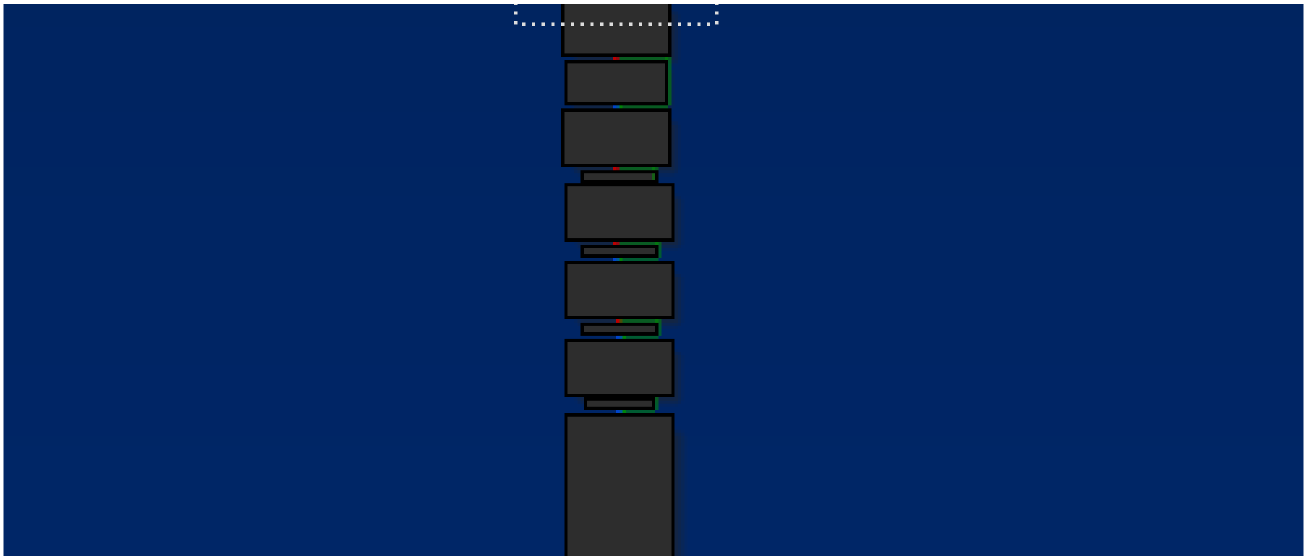




```
if ( (unsigned __int8)std::operator==<char>(switch_command, "499") )
{
  *(_DWORD *)&hostshort = unsigned4_1;
  hThread_4 = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)mw_writesCNCfromRegKey,
  ResumeThread(hThread_4);
  SetThreadPriority(hThread_4, 0);
  unsigned4_1 = unsigned4;
}
if ( (unsigned __int8)std::operator==<char>(switch_command, "599") )
{
  *(_DWORD *)&dword_436248 = unsigned4_1;
  ::hThread_2 = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)mw_599__, 0, 4u, &add
  ResumeThread(::hThread_2);
  SetThreadPriority(::hThread_2, 0);
}
if ( (unsigned __int8)std::operator==<char>(switch_command, "999") )// change C2
{
  *(_DWORD *)byte_4360E4 = 1;
  strncpy(cp, Source, 0x10u);
}
if ( (unsigned __int8)std::operator==<char>(switch_command, "555") )
{
  strncpy(cp, Source, 0x10u);
  mw_THIS_is_where_C2_is_defined(1);
}
if ( (unsigned __int8)std::operator==<char>(switch_command, "315") )
  set_to_0_by_315 = 0;                         // zero this out
if ( (unsigned __int8)std::operator==<char>(switch_command, "312") )
  set_to_0_by_312 = 0;                         // zero this out
if ( (unsigned __int8)std::operator==<char>(switch_command, "313") )
  set_to_0_by_313 = 0;                         // zero this out
if ( (unsigned __int8)std::operator==<char>(switch_command, "666") )
  set_to_1_by_666 = 1;                         // set this to 1
```







**[Updated 04.28.2020]:** Thanks to @maciekkotowicz's great work, we now know that EYservice is in fact a passive backdoor with no hardcoded infrastructure. The backdoor is listening for UDP packets on port '1234' and allows for a ping response, victim info request, or file download. For further details, please refer to the MalwareLab.pl blog.

### The Subcomponent DLLs

Subcomponent DLLs include multiple abused resources as well as a couple of seemingly custom libraries. The former include the common LAME MP3 encoding library (UPX packed) as well as a more obscure bitmap library. These are abused to implement hot mic and screengrab features, respectively. Another subcomponent is the 'hodl.dll' (internally named 'keydll3.dll') library used for keylogging. This appears to be a more common keylogger but that claim could use further scrutiny.

Finally, the custom libraries are 'godown.dll' (our original indicator) as well as 'filesystem.dll'. Both are treated as type libraries and registered as OLE controls. The Filesystem library includes functionality to enumerate attached drives and traverse folder structures. The GoDown library is used for system shutdown. **[Updated 04.28.2020]**

For a more comprehensive breakdown of these components, refer to the Checkpoint Research blogpost **[Updated 05.05.2020].**

### A Further Oddity – The MicroOlap Packet Sniffer

A core function of EYService includes a further drop, a packet sniffer. The orchestrator will unpack and drop a kernel driver (pssdk41.vxd, pssdk41.sys) used to sniff packets from the victim machine's interfaces. The packets are then parsed looking for something in particular. Perhaps this allows for a sneaky means of command-and-control or more sophisticated uses. At this time, I've not determined what it's parsing in particular.

Interestingly, the packet sniffer is also referenced in the EQGRP drv_list.txt. Other versions are also referenced, as shown in the image below:

```
"pssdk31","*** microOLAP Packet Sniffer SDK Driver ***"
"pssdk40","*** microOLAP Packet Sniffer SDK Driver ***"
"pssdk41","*** microOLAP Packet Sniffer SDK Driver ***"
"pssdk42","*** microOLAP Packet Sniffer SDK Driver ***"
"pssdklbf","*** microOLAP Packet Sniffer SDK Driver ***"
```

Interestingly, focusing on the alternate filenames brought up an earlier version of this Nazar orchestrator (sha256: 1c02043ca00d087f1aac0337f89bf205985e1f20641bf043c9b7b99e0c9dc002). This earlier version drops 'pssdk31.drv' instead of the 4.1 version mentioned above.

### Avenues for Further Research

SIG37 has proven a rewarding mystery, unearthing a previously undiscovered subset of activity worthy of our attention. Apart from several places where more skilled reverse engineers can contribute to better understanding the samples already discovered, there's an opportunity for threat hunters with access to diverse data sets and systems to figure out just how big this iceberg really is. Are we looking at an internal surveillance framework? Is this part of an already known cluster of activity? Or can we add another predatory animal to our overpopulated zoo?

Happy Hunting!

---

## Appendix – Technical Indicators

### Nazar Hashes

#### gpUpdates.exe

- 4d0ab3951df93589a874192569cac88f7107f595600e274f52e2b75f68593bca
- d34a996826ea5a028f5b4713c797247913f036ca0063cc4c18d8b04736fa0b65
- eb705459c2b37fba5747c73ce4870497aa1d4de22c97aaea4af38cdc899b51d3
- d9801b4da1dbc5264e83029abb93e800d3c9971c650ecc2df5f85bcc10c7bd61

#### Unnamed Droppers

- 75e4d73252c753cd8e177820eb261cd72fecd7360cc8ec3feeab7bd129c01ff6

- 1110c3e34b6bbaadc5082fabbdd69f492f3b1480724b879a3df0035ff487fd6f

**Distribute.exe**

6b8ea9a156d495ec089710710ce3f4b1e19251c1d0e5b2c21bbeeab05e7b331f

**svchost.exe**

2fe9b76496a9480273357b6d35c012809bfa3ae8976813a7f5f4959402e3fbb6

**Filesystem.dll**

1110c3e34b6bbaadc5082fabbdd69f492f3b1480724b879a3df0035ff487fd6f

**Godown.dll**

- 967ac245e8429e3b725463a5c4c42fbdf98385ee6f25254e48b9492df21f2d0b
- 8fb9a22b20a338d90c7ceb9424d079a61ca7ccb7f78ffb7d74d2f403ae9fbeec

**hodll.dll**

0c09fedc5c74f90883cd3256a181d03e4376d13676c1fe266dbd04778a929198

### Abused Common Resources

---

**pssdk41.sys**

048208864c793a670159723b38c3ea1474ccc62e06b90833bdf1683b8026e12f

**ViewScreen.dll**

5a924dec60c623cf73f5b8505e11512ad85e62ac571a840ab0ff48d4a04b60de

**lame_enc.dll**

- c84100d52c09703e32951444bd7ba4e22c5d41193e7420aacbbc1f736f4c4e1f
- 0091e2101f00751c4020ef8e115cfe12a284c9abacc886f549b40a62574a7510

---

YARA rules available here

J A G-S