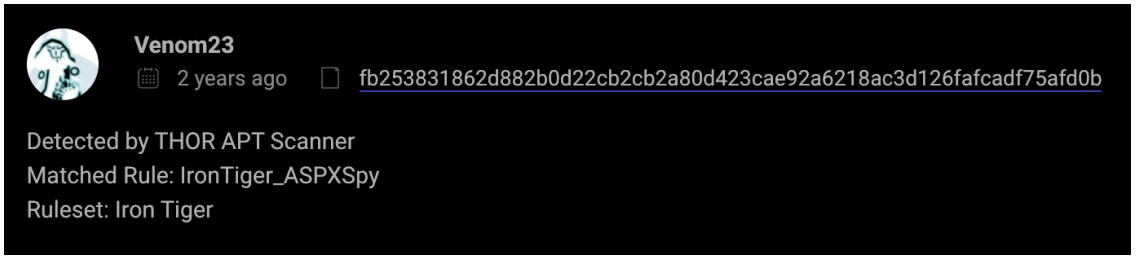
Nazar: A Lost Amulet — The Lost Reports

By Apr 22 Written By J A G-S

Published: 2001-04-22 · Archived: 2026-04-05 13:02:04 UTC



Automated community comment with the misleading detection

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

SIG37 function from 'sigs.py'

'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

[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:
