The “Hikit” Rootkit: Advanced and Persistent Attack Techniques (Part 2)

By by Christopher Glyer

Published: 2012-08-22 · Archived: 2026-04-06 00:22:40 UTC

By Christopher Glyer and Ryan Kazanciyan

In the first part of this series we introduced the "Hikit" rootkit and discussed some of its distinctive characteristics, particularly the clever mechanisms it uses to load on a compromised system. Today, we'll take a look at some of the counter-forensic techniques that it utilizes to stay hidden in a compromised environment, provide an Indicator of Compromise (IOC) used to find host-based evidence of the malware, and discuss how attackers took advantage of its functionality.

Signed and Delivered

When conducting triage analysis to identify unknown malware on a system, investigators often whitelist valid binaries by comparing their MD5 hashes against "known good" lists such as the NIST Software Reference Library. Similarly, eliminating files that have been digitally signed can be another effective data reduction technique - but assumes a certain level of risk, depending on which certificates you choose to trust.

Malware authors are well aware that digitally signed binaries are more likely to be overlooked (and may even be necessary if attempting to install drivers on certain versions of Windows or conducting other automated operations where some degree of certificate validation occurs). Stuxnet and Flame are both examples of sophisticated malware that used digital certificates, albeit from very different sources, to help achieve their objectives.

So how does this relate back to "Hikit"? During installation, "oci.dll" extracts a number of files from its resources section:

- The rootkit driver "W7fw.sys"
- Several requisite .INF and .CAT files for the driver
- A digital certificate "GlobalSign.cer", along with a copy of Microsoft's Certificate Manager tool "certmgr.exe"

The attacker self-generated "GlobalSign.cer" to masquerade a legitimate certificate issued by GlobalSign - it was *not* stolen nor legitimate. The malware proceeds to use "certmgr.exe" to install the certificate to the local trust store as a root CA and Trusted Publisher using the following two commands:

- certmgr.exe -add GlobalSign.cer -c -s -r localMachine Root
- certmgr.exe -add GlobalSign.cer -c -s -r localMachineTrustedPublisher

It then attempts to disable driver signing verification by tampering with several registry keys. Finally, it completes the driver installation process and checks that it is properly loaded.

We now have enough information to build an effective host-based IOC, an example of which is shown in the figure below, that can reliably identify this malware. The IOC includes simple traits like unique filenames, process handles, MD5 hashes, service configuration, and the invalid digital signature information. It also has a more complex set of expressions to identify suspicious instances of "oci.dll" based on combinations of imported functions, resource names, version information metadata, and compilation time.

Figure 1: Indicator of Compromise (IOC) for "Hikit" rootkit

"Hikit" Put to Use

So what does the rootkit driver do? "Hikit" uses an interesting covert mechanism for command and control. It installs itself as a virtual network adapter layered between the NIC and overlying protocol drivers. This allows it to covertly monitor incoming packets, intercept command and control data as it enters the network stack, and then

spawn user-mode threads to parse them accordingly. (And in case you're wondering, we're definitely glossing over the technical details that our awesome malware analysts uncovered during reverse engineering).

But this should have you wondering about the context in which such a backdoor would be useful. After all, most internal servers and workstations in a corporate environment are going to be behind several layers of firewall and NAT'ing - that's why most backdoors communicate *out*? So why go through all of this trouble for a rootkit that can only listen for inbound C2?

The pattern was obvious once we saw *which* systems were infected with the rootkit: nearly all of them were Windows servers residing in the victim's DMZ and running web services on ports 80 and 443. The attacker was thereby able to covertly issue C2 over these same inbound ports using the backdoor client. Without knowing exactly what C2 signatures to look for, most network monitoring solutions would infer the traffic to be no different than other incoming HTTP sessions.

Furthermore, in addition to the typical set of backdoor features (launch a shell, run commands, transfer files, etc.), "Hikit" can force infected systems to operate as an open proxy. When coupled with the fact that several internal firewalls were more permissive than necessary, this allowed the attacker to tunnel Remote Desktop and obtain interactive access to internal systems, authenticating to each using previously compromised credentials. This also allowed them to create "hop points" among internal and external network segments by placing the rootkit (or rootkit client) on hosts that were multi-homed.

Malware like "Hikit" reinforces an important lesson that digital forensic analysts should all take to heart: always question your assumptions and be prepared to expect the unexpected. Rapidly evolving tools and techniques will increasingly test truisms like "You can always trust signed binaries" or "Modern backdoors are always going to initiate outbound C2". Following your leads, generating effective IOCs, and fully scoping enterprise-scale incidents can help ensure you stay a step ahead of a sophisticated adversary.
