MALWARE ANALYSIS 1

THE MISSING LNK

Peter Ferrie
Microsoft, USA

LNK files are everywhere in *Windows*, so ubiquitous that they are rarely even recognized for what they are: complex structures containing pointers to Portable Executable files and, ultimately, executable code.

Some of the icons that appear in the Control Panel are visible because of LNK files. Many of the entries in the Start Menu and on the Desktop are LNK files. In most cases, the LNK references a file, and specifies an icon to display. When an application is used to view the LNK file, such as browsing a folder using *Windows Explorer*, the *Windows* shell parses the format and determines what to display. LNKs are not limited to just files, though. They can be shortcuts to drives such as a shared network location or a floppy disk (as used by the 'Send To' menu, for example). The 'Recent File List' in *Microsoft Office 2007* applications is composed of LNK files.

Overall, LNK files do not pose a direct threat. Of course, some LNK files can point to malicious executables that run when the LNK file is clicked, and some LNK files can point to harmless files and yet still perform malicious actions (such as when the command prompt is executed, but given the instructions to delete some files). Some LNK files can themselves be malicious by virtue of their contents (such as the self-executing LNK file virus from several years ago, where the LNK file carried an actual Portable Executable file, and executed it in a rather roundabout fashion). Then there are the LNK files produced by W32/Stuxnet, which allow the execution of arbitrary code without the need for any user interaction (other than browsing to a folder that contains such a file, with some further clarification below).

LNKS TO THE PAST

The LNK file format has existed since the days of *Windows 95*, but it was not documented publicly by *Microsoft* until 2009, and even then it was incomplete and interrupted. After the Stuxnet malware caused quite some interest in the LNK file format, the documentation was temporarily removed from the *Microsoft* website, and then re-released as 'new'. However, the only difference between the two versions is some formatting, and the dates of referenced external files. The two versions are equally useless as far as determining how to parse the LNK files produced by Stuxnet goes, because the section that is being exploited is not documented in either version.

So, in order to understand what Stuxnet did with LNK files, we first have to understand what is inside a LNK file.

LNK LAYER

A LNK file begins with a 0x4c-byte-long header. The first field of the header is the 'HeaderSize', which specifies the size of the header, including itself. It must contain the value 0x4c. The next field is the 'LinkCLSID' (though it resolves to a registry key called 'Shortcut'). The CLSID must be '{00021401-0000-0000-C000-000000000046}'. That's the extent of the constant bytes for the header, as far as Stuxnet-style files go.

There is a 'LinkFlags' field, which is supposed to specify information about the type of link and the presence of optional structures. Unfortunately, all but two of the flags can be set arbitrarily, despite the corresponding structures being missing from the file. Only one bit is required to be set, and that is the HasLinkTargetIDList. When the bit is set, it specifies that the file is saved with an item ID list, and the corresponding 'LinkTargetIDList' structure follows the ShellLinkHeader structure immediately.

One important-sounding bit in the LinkFlags field is the 'IsUnicode' bit. This is supposed to be set if a file contains Unicode strings, and the documentation states that the bit 'SHOULD' (in upper case in the documentation) be set. However, the 'should' apparently refers to the fact that LNK files should be in Unicode format, as opposed to ANSI format. Unfortunately, the bit is entirely useless because a file can have the bit set and still be in ANSI format. Alternatively, the bit can be clear and yet the file can still be in Unicode format. The way to determine the format of the strings will be described below.

The two bits that cannot be set arbitrarily are 'HasExpString' and 'HasDarwinID'. If either bit is set, then the corresponding structure must be present in the file, but the presence of either of them prevents the use of the structure that the Stuxnet LNK files require.

CUFF LNKS

The LinkTargetIDList structure is an 'IDList' structure which contains a collection of 'ItemID' structures. In the case of a Stuxnet LNK file, there are three ItemID structures. Each of these contains a size field, a type field, and a data field.

The first ItemID structure in a Stuxnet LNK file contains the type and CLSID for the 'My Computer' or 'Computer' element (the specific name depends on the version of *Windows*, but the name is not relevant). The type field is two bytes long, but only one of those bytes is checked by *Windows*. The type must be 0x1f, and the CLSID must be '{20D04FE0-3AEA-1069-A2D8-08002B30309D}'. Although the value in the size field is most commonly 0x14, the size of the structure is not a constant. The size of the structure can be increased in a hand-crafted file (though Stuxnet does not do this), by adding additional data after the CLSID field. Then the value in the size field can be increased accordingly. This alteration is known to break at least one publicly available scanning tool. I attempted to contact the author of the tool regarding this attack, but received no reply, and the tool remains unchanged.

The second ItemID structure in a Stuxnet LNK file contains the type and CLSID for the 'Control Panel' or 'All Control Panel Items' element (again, the specific name depends on the version of *Windows*, but is not relevant). As before, the type field is two bytes long, but only one of those bytes is checked by *Windows*. The type must be 0x2e, and the CLSID must be '{21EC2020-3AEA-1069-A2DD-08002B30309D}'. Again, the size of the structure is not constant. The presence of the Control Panel CLSID leads us to the flaw that Stuxnet exploits.

The third ItemID structure in a Stuxnet LNK file contains an undocumented (to the point that I don't even know the correct name) 'Control Panel applet' structure, which contains a size field, an icon index, some truly 'spare' fields, and a path to the file that holds the icon to display. This is where the magic happens. When the Control Panel is displayed, *Windows* queries the corresponding LNK files for the icons to display. Based on the behaviour of the patch (see below), this was intended to apply only to registered Control Panel applets (that is, the registry key 'Software\Microsoft\Windows\CurrentVersion\Control Panel\Cpls', under either 'HKCU' or 'HKLM', contains subkeys or values that name them), and ideally they would have been placed in the '%windir%\system32' directory. Unfortunately, neither of these conditions was enforced, allowing LNK files to access files in arbitrary locations in any context whereby an icon would be displayed. This is why we can reproduce the problem by simply 'browsing a folder using *Windows Explorer*'. The referenced files are *Windows* DLLs, usually with a suffix of '.CPL', and *Windows* will load them in order to retrieve the address of an exported function which is used to display the icon. Of course, in the case of Stuxnet, control is gained when *Windows* loads the file, and thus no exported function is necessary. Further, no warning is given when the named file is loaded.

To complicate matters further, the Control Panel applet structure comes in two forms, as noted above: Unicode and ANSI. The correct format can be determined by examining the values in particular locations within the structure. For the Unicode format, the six bytes beginning at offset 8 must be '00 00 00 00 00 6A'. Ten bytes later is the path in Unicode characters. This is the only format that Stuxnet LNK files use. For the ANSI format, the four bytes beginning at offset 8 can have almost any value (see below), including all zeroes, and which, if the following two bytes are not checked, might lead someone to assume that the Unicode format is in use. For the ANSI format, four bytes later is the path in ANSI characters.

D-LNKS

As far as the path goes, the typical case is to have a drive letter, directory and filename. Of course, this ties the file to a specific drive configuration. That would be fine if the drive letter were constant, but in the case of Stuxnet, there is no guarantee of that. Specifically, Stuxnet places LNK files on USB removable storage media (among other places). Since the drive letter is assigned dynamically, because the order of device insertion can vary, Stuxnet needed a way to refer to the file regardless of the drive letter. This was achieved by querying the registry for the hardware ID. The result looks like '\\.\STORAGE#RemovableMedia#7&xxxxxxxx&0&RM#{53f5630d-b6bf-11d0-94f2-00a0c91efb8b}\<file>', where the 'x's are a unique value specific to the device, and the CLSID is the Device Class GUID for a volume.

Further, the path does not need to look like either of these examples. The path can be in UNC form, allowing a file to be loaded from a remote location, such as a network share within a corporation. However, since the WebDAV redirector is also running by default, the location can be very remote. That is, anywhere on the Internet. This variation is used by the exploit module in the Metasploit framework.

WEAK LNKS

One virus writer posted a message to a forum claiming that anti-virus researchers 'read docs instead of code' and that his post contained the '*real* format for LNK files'. While his description was mostly correct (even including the two bits that cannot be set, though he did not name them, and did not describe why they cannot be used), and he did find something that I did not know (the use of a relative path without a drive letter was not supported by my detection at the time), he did manage to get two things wrong. The two things that were wrong result in essentially 'random' execution of the LNK files. Perhaps he didn't read the code carefully enough. Or perhaps that was the intended result.

The first wrong thing relates to the 'unused' bytes that live in the field before the path to the file. Two of those bytes are actually a separate field that is supposed to contain the size of the path. This is used as a relative pointer to the description text, and it is used by *Windows*. If the field contains a value that is too large (that is, beyond the end of the LNK file), then an exception might occur while *Windows* attempts to copy the description string. In that case, the LNK file will not be parsed any further. That might sound potentially interesting to an attacker, since obviously there is no check that the pointer is valid. However, what lies beyond the end of the file is essentially random data, and the string copy operation has a limited length, so it cannot be exploited. The value is not supposed to be zero, either, since that would cause the filename to become the 'description', but such a situation causes no ill effects.

The second thing the virus writer got wrong relates to the non-terminated string. If the termination bytes are removed, then when the LNK file is loaded, whatever happened to be in memory at the corresponding location will be used instead. The bytes there might not be zero, resulting in a string that appears to be longer than it was supposed to be. Again, that might sound potentially interesting to an attacker, but as before, what lies beyond the end of the file is essentially random data, and the string copy operation has a limited length, so it cannot be exploited. A further result is that the check for the existence of the file is likely to fail, since those random bytes will become part of the filename that is examined.

STRONG LNKS

Microsoft patched the behaviour to check for CPL files in several locations, and a list is made of the files found for use later. The locations are the 'MMCPL' key in 'control.ini' (which is redirected to the registry, but the location is not constant), '%windir%\system32' and 'Software\Microsoft\Windows\CurrentVersion\Control Panel\CPLs' under both 'HKLM' and 'HKCU'. Any CPL file that is found is excluded if it is marked as 'don't load' according to its registry key.

Once the list has been made, the file referenced by the LNK file is checked against each entry in the list. If the file is not in the list, then *Windows* makes a change to the loading method, to force the use of a default system icon instead of the requested icon. This also prevents the referenced file from being loaded. Internally, the path is converted to the form '<path>,<icon>,<description>', where <icon> is zero in Stuxnet LNK files. However, if the file is not found in the list, then the path is converted to '<path>,-1,<description>'. Prior to creating the list, the original path is checked for the presence of a comma, in order to prevent a path that has the internal form prior to conversion, since that could have been used to defeat the other checks in the patch.

CONCLUSION

So now we know what the Stuxnet LNK files do. As for what else Stuxnet does, the details would fill a conference paper (or two!). It's too late to submit a paper for the *Virus Bulletin* conference this year, but maybe I'll be first in line when the call for papers opens for VB2011.