MALWARE ANALYSIS 2

LIKE A BAT OUT OF HELL

Peter Ferrie
Microsoft, USA

A polymorphic batch file seems like a holy grail to some virus writers, perhaps because of how insanely difficult it is to produce one. In spite (or perhaps because) of the challenges, a virus writer has managed it with BAT/Lymer.

BACK TO BASICS

The virus begins by checking the first parameter that was used to run the program. If it is not a special string (a variation of the virus writer's name), then the virus will create a new console window and run the virus there by passing the special string. The second console window is minimized. This allows the virus to run in what should be the background, and the host code to run immediately. The virus writer calls this technique 'stealth' execution. It seems to be the first time that the technique has been used for such a purpose. However, the way in which the virus runs itself might be considered a bug. The virus does not specify a priority class when creating the second console window. As a result, the virus runs with the same priority as the original process.

The virus attempts to enable a command extension that was introduced in *Windows 2000* (despite several references that state incorrectly that the changes were introduced in *Windows XP*). There is no check that this was successful. However, there are only two ways in which it can fail. The first is that the platform is simply too old (i.e. *Windows 95* or *Windows NT*). Secondly, it can fail if the extension is disabled. This can be achieved in two ways. First, the command processor can be launched with the '/V:OFF' switch. This is a local change that affects only that copy of the process. The extension can also be disabled if the 'Software\Microsoft\Command Processor\DelayedExpansion' value in either the HKCU or HKLM hive is set to zero. This is a global change that affects all processes.

The code used to check whether the extensions are enabled is something like this:

```
verify r 2> nul
setlocal enableextensions
if errorlevel 1 goto :eof
```

The 'verify' line will ensure that the error level is set to zero. The 'setlocal' line will set the error code only if it fails to enable the extensions. If the error level is non-zero, then the code will exit.

%RANDOM TIME%

The virus retrieves the current time by writing the output of the 'time' command to a file, reading it back, extracting the minutes field, and then placing the result in an environment variable. It is not known why the virus writer didn't simply use the '%time%' internal variable directly. The time value is used to seed the random number generator in the virus, which is a copy of the *Microsoft Visual C* random number generator ported to the batch language. It is not known why the virus writer didn't use the '%random%' internal variable instead.

VARIABLE VARIABLES

The virus places the name of each variable that it uses into a pseudo-array (including the name of the pseudo-array itself). Once that has been done, the virus constructs a new, randomly generated name for each entry. The names are between eight and 11 letters long, and the case of each letter is chosen randomly. The code that selects the random letter uses an unusual optimization. Normally, a virus would choose a random number in the range of one to 26, to correspond to the letters 'A' to 'Z', and then convert to lower case if that is the chosen mode. However, this virus chooses a random number in the range of zero to 63, and uses that value as an index into a string. The string consists of the letters 'A' to 'Z' and 'a' to 'z', as expected, but some additional characters are appended after each alphabet in order to increase the length of the sequences to 32 characters. This is necessary because an attempt to access a value beyond the end of a string will return a null character, which could result in a variable with no name if the null is the first character. Since the mask is larger than the size of the alphabet, the characters in the two padding strings will be used occasionally, resulting in certain characters appearing slightly more often than others. There is also one space at the very end of the string, the reason for which will be described below. The new names are used when the virus constructs a new representation of itself.

TOKEN GESTURE

In order to create a new representation of itself, the virus reads each line of virus code from the infected file, and then writes it to a temporary file. The virus stops parsing after it writes the line that contains the special string that is used as the parameter when starting the virus. After the virus has been extracted from the infected file, it reads each line from the temporary file, tokenizes it, and then parses the content. The virus knows how to interpret every component of every keyword that it uses, and it could rebuild itself entirely if only the batch tokenizer would cooperate.

Unfortunately for the virus writer, it does not cooperate. The way in which the virus reads the virus code results in the replacement of variables with their values in many cases, and these values are written to the temporary file. To work around this, the virus uses specially encoded 'rem' lines in the appropriate locations, to describe the format of the original line. These all begin with the '_' character, followed by the text that should replace the value. The location of the value depends on the line that is being parsed. For example, the 'if' lines will have a value inserted prior to any '~' character. A 'set' line will have a value inserted after the '=' character.

Since certain characters are considered to be 'special' in batch files, they cannot be placed directly anywhere in the code, including in the rem line itself. As a result, the virus has to use an encoded form to represent them. The encoded forms begin with a '#' character, followed by a single letter that represents the special character. The virus uses the letter 'p' to represent the '%' character, 'x' to represent the '!' character, the letter 't' to represent the '^' character, and the letter 'c' to represent the ':' character (although in this case, the letter 'c' is not checked, so any unused character could appear here).

SIZE DOES MATTER

The virus appears to have been optimized for small size (ignoring the 'time' technique above), making the code very dense and quite difficult to read in some places. The minimum number of characters are checked when comparing strings, including using an index into the string in order to select a unique character instead of comparing multiple leading characters. However, the virus writer appears to have overlooked the fact that when accessing a substring beginning with the first character, the index is not needed. For example, the line

```
if /i "!_atok:~0,4!" == "echo" (
```

could have been written as

```
if /i "!_atok:~,4!" == "echo" (
```

There are many lines like this. There is also a 'rem _#p1' line which would decode to '%1', however the line that follows does not contain any reference to a '%1'. Given the line that follows, the 'rem' line indicates that the function originally received its parameter in a different way. It has no effect on the behaviour of the virus in its current form because the line that follows does not require an encoded 'rem' line. However, if someone were to add a line that does require an encoded 'rem' line at that location, then the line would be replaced in an incorrect way.

IF YOU BUILD IT...

The virus produces one polymorphic representation of itself per run, and uses that representation to infect all files that it can find. This makes it a slow polymorph. Each run can take upwards of ten minutes to produce a new copy – which makes it a very slow polymorph. The polymorphism has three forms.

The first form is a random number of spaces, from one to four, before, between, and after every token. This is where the trailing space from the alphabet string is used. Since the tokenizer considers a space to be a delimiter, the virus cannot embed a space in an encoded 'rem' line for that purpose. This is because the line will appear to have two tokens instead of one. A 'rem' line with two tokens has a special meaning for the virus code, and the space character still cannot be used in that case. The virus also cannot write a line that ends in a literal space to the temporary file, because the tokenizer will strip the space before writing the line. The solution that the virus uses is to assign a line that ends in a space to an environment variable, and use an index into the alphabet string to read and write the space character indirectly.

The second form is a random mapping of letter case. Since batch files are essentially case-insensitive, this allows for a lot of flexibility in appearance. The one exception to that rule is for 'if' statements, but the '/i' switch enables case-insensitivity there, too. The 'if' statements are treated in a special way by the virus. If the text to compare is entirely alphabetic, then the virus uses the encoded 'rem' line, with a second token that matches the text, to indicate that the text in the 'if' statement can have its case mapped randomly. The second token in the encoded 'rem' line will have its case mapped randomly, too.

The third form is the insertion of random 'rem' lines. These lines do not begin with the '_' character, so the virus can identify them easily and ignore them when extracting the virus code. The virus will produce a random number of 'rem' lines, from zero to three, after each line of virus code. Each of those lines will contain a random number of components, also from zero to three. Each of the components will be from zero to seven letters long. The case of each of the letters is chosen randomly.

There is an implicit fourth form of polymorphism, the description of which was begun above. Each variable name in the virus code is replaced by a randomly chosen name. The virus achieves this by searching each line of virus code for each of the variable names.

After the new representation has been created, the virus searches within the current directory for all files whose suffix is 'bat'. If the sum of the file size and the virus size is less than 60,000 bytes, and if the file is not infected already, then the virus will attempt to prepend itself to the file. The infection marker is for the second and third characters of the first line in the file to be 'if'. This is intended to match '@if', but in a way that allows a random case mapping. The virus does not pay attention to the file attributes (perhaps because that would require the use of an external program, and then the virus would no longer be ‘pure batch’), so a file will not be infected if it has the read-only attribute set.

LYME DISEASE

The virus has a fatal bug when run under *Windows XP*: the line ‘set _out=!_out!%%~’, which is supposed to append ‘%~’ (the double ‘%’ is required in order to emit a single ‘%’), does not append anything. It is not known why this happens, but it appears that a line cannot end with that sequence of special characters. The bug appears to be in *Windows*, not in the virus. If an additional character is added to the line, then all of the characters are appended correctly. If the virus had added that additional character, and then removed it after the characters were appended, then the virus would work on *Windows XP*, too. The bug causes the virus to fail to parse anything, and then to delete itself, because there is no new representation.

The virus has an ‘even more’ fatal bug when run under *Windows 2000* (the bug that exists in the *Windows XP* command processor is present here, too). The line ‘set /a _val1 += “_ind”’, which is supposed to select the case of the randomly selected letter, does not make use of the ‘_ind’ variable. Instead, the value is always treated as a zero. This might be considered to be a bug in *Windows*, rather than in the virus, however the behaviour is undefined because the documentation regarding the use of quotes is ambiguous regarding this situation. The virus contains another line in the same style, but without the quotes, so we can assume that this is a bug in the virus. If the quotes were removed, and if the fix were applied as for the *Windows XP* case, then the virus would work on *Windows 2000*, too. The bug causes the virus to emit strings that are composed solely of the letter ‘A’.

The virus works correctly on *Windows 7* without modification. This is especially interesting, because the virus writer is known for producing very compatible code. For example, most of his binary viruses still support *Windows 95*. His more recent viruses ‘merely’ require *Windows NT*. It is clear that he did not test this virus on anything other than a relatively recent platform such as *Windows Vista* (assuming that it works there – I did not try it) or *Windows 7*. Perhaps he finally upgraded his machine.

CONCLUSION

It’s clear that some people have too much time on their hands, to have found a way around all of the limitations and quirks of the batch language, and produced a virus like this. However, if we can’t stop them from writing viruses at all, then we can at least be thankful that they’re not writing something much worse than this.