VIRUS ANALYSIS

LIONS AND TIGRAAS

Peter Ferrie
Symantec Security Response, USA

Texas Instruments makes very advanced graphing calculators. What is particularly interesting is that these calculators are programmable, and the resulting applications can be stored in the memory of the calculator. The applications can also be altered by other applications, and that means they can be infected by viruses. This brings us to TIOS/Tigraa, which is the second known calculator virus (the first having been TIOS/Divo), though Tigraa is the more interesting of the two.

MEMORY RESIDENT

When the virus is first executed, it checks whether the ROM call table is in ROM (at 8Mb or greater) or RAM (below 8Mb). This serves as the memory infection marker. If the call table is in ROM, then the virus allocates some memory for it and copies the call table to the allocated memory. The virus uses a function to allocate memory at the top of the heap, then another function to de-reference the returned handle to get the memory address. This is despite there being a single function that performs both of these actions.

In the same way, the virus also allocates some memory to hold a copy of the virus body itself, allowing it to stay resident in memory after the host application terminates. The virus hooks the SymFindNext() function in the new call table, then replaces the original call table pointer with a pointer to the new call table.

PAYLOAD

The virus only checks whether the payload should run when an infected application passes control to the virus code. The payload itself activates only if one of the system timers contains the value 119. However, since that timer is updated at a rate of about 1,500 times per second by default, actually seeing the payload is likely to be a rare event.

The payload is very simple: it clears the screen and prints the text 't89.GAARA ' at column 55 of the first row, then returns control to the host. Despite the message, which suggests the virus may be specific to the *TI89* calculator, the virus also runs on the *TI92+* and *Voyage 200* calculators. Older calculators, such as the *TI84*, use a different CPU (*ZiLOG Z80*), and the virus does not work on them.

SYMFINDNEXT & INFECTION

The SymFindNext() function is a symbol enumeration function. SymFindFirst() begins the search, and the two functions are fairly similar to the FindFirstFile() and FindNextFile() functions on *Windows*, for example.

When an application executes SymFindNext(), the virus gains control. It executes the original SymFindNext() in order to get the information, and the contents of the returned entry will be examined for the possibility of infection. A symbol is a candidate for infection if it is a file (as opposed to a folder), that is not a twin symbol, archived (because the Flash memory is protected), locked, or deleted.

The virus checks that the file is not infected already by searching forwards through the entire file for the string 'GAA'. It is unclear why the virus author did it that way, given that the file size is known, and it would be trivial to search backwards instead – especially since the string appears near the end of infected files. Only after searching for the infection marker does the virus check whether the file is an assembly file, which is the only infectable file type.

Once a suitable file has been found to infect, the virus moves the relocation table further down in the file to make room for the virus body. The reason for this is that the relocation table must appear after the image. It seems that the virus author did not realise that the relocation table is always properly aligned, so copying it can be done in just two instructions instead of 11.

This is the point at which this virus really differs from TIOS/Divo. Whereas Divo replaced the first instruction in the file with a branch to the virus body, Tigraa searches the file for the first instance of a specific instruction sequence ('unlk a6/rts'). If that sequence is found, then Tigraa will replace it with a branch to the virus body. Once again, the virus author appears to not realise that since all *Motorola 68000* instructions are 16 bits long, the instruction will always be properly aligned, so the store can be shortened by one instruction. If the sequence is not found, the file will still contain the virus body, but the virus will never gain control.

CONCLUSION

This virus appears to have been written by a beginner to *Motorola 68000* assembler programming, based on the use of 'lea/mov -(sp)' instead of 'pea', a test for zero after an 'and', the apparent inability to decide between 'clr' and 'sub', and so on. Such things would be forgivable in a true beginner, but the virus author is a well-known security researcher. He released the virus source code under his own name, Piotr Bania, rather than one of his aliases ('Lord Yup' and 'dis69'). It's funny, in a way, that the first virus that we know he wrote came only after he left the 29A virus-writing group. In any case, it's still a virus, and that is to be condemned. Just because you *can* write one doesn't mean that you should.